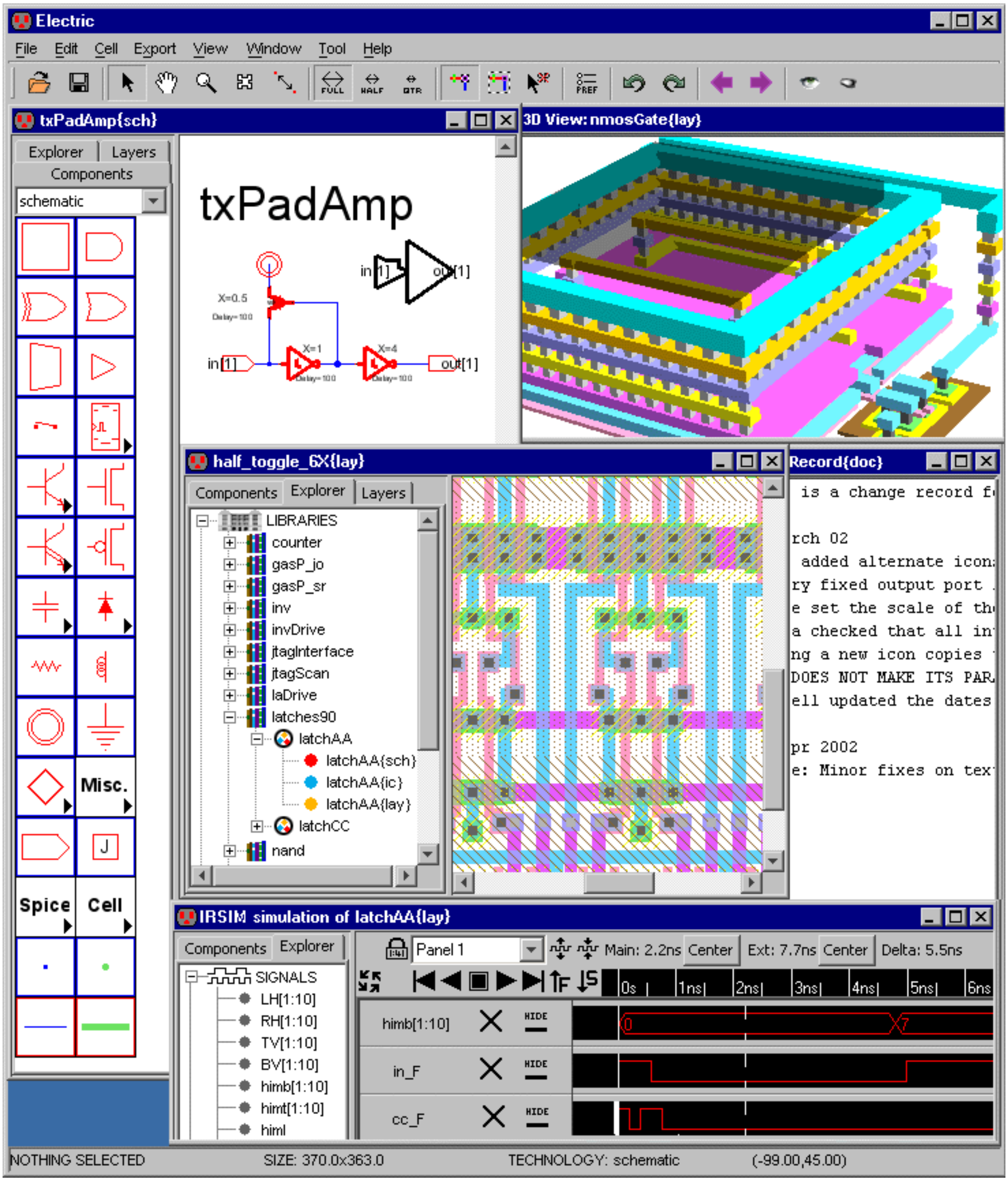


```
 is a change record f

rch 02
 added alternate icon
ry fixed output port
e set the scale of th
a checked that all in
ng a new icon copies
DOES NOT MAKE ITS PAR
ell updated the dates

pr 2002
e: Minor fixes on tex
```

Using The Electric VLSI Design System

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Chapter 1: Introduction

1−1: Welcome

Now you have it!

A state−of−the−art computer−aided design system for VLSI circuit design.

Electric designs MOS and bipolar integrated circuits, printed−circuit−boards, or any type of circuit you choose. It has many editing styles including layout, schematics, artwork, and architectural specifications.

A large set of tools is available including design−rule checkers, simulators, routers, layout generators, and more.

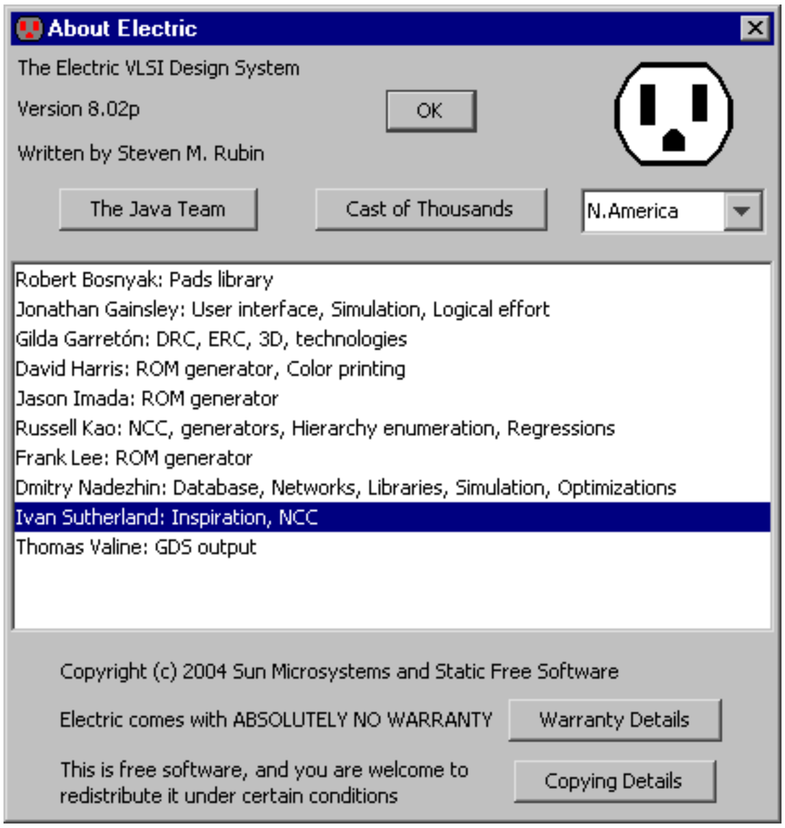
Electric interfaces to most popular CAD specifications including VHDL, CIF and GDS II.

The most valuable aspect of Electric is its layout−constraint system, which enables top−down design by enforcing consistency of connections.

This manual explains the concepts and commands necessary to use Electric. It begins with essential features and builds on them to explain all aspects of the system. As with any computer system manual, the reader is encouraged to have a machine handy and to try out each operation.

1−2: About Electric

The **About Electric...** command (in menu **Help**) shows you the names of the Electric development team. It also outlines you legal rights with respect to Electric.



About Electric

The Electric VLSI Design System

Version 8.02p

Written by Steven M. Rubin

Robert Bosnyak: Pads library
Jonathan Gainsley: User interface, Simulation, Logical effort
Gilda Garretón: DRC, ERC, 3D, technologies
David Harris: ROM generator, Color printing
Jason Imada: ROM generator
Russell Kao: NCC, generators, Hierarchy enumeration, Regressions
Frank Lee: ROM generator
Dmitry Nadezhin: Database, Networks, Libraries, Simulation, Optimizations
Ivan Sutherland: Inspiration, NCC
Thomas Valine: GDS output

Copyright (c) 2004 Sun Microsystems and Static Free Software

Electric comes with ABSOLUTELY NO WARRANTY

This is free software, and you are welcome to redistribute it under certain conditions

This manual is available while running Electric. Use the **User's Manual...** command (in menu **Help**) to see this manual (you may already be doing that).

While inside of the manual, click "Menu Help" to get help with Electric's pulldown menus. It displays a pulldown menu inside of the manual page which mimics the real pulldown menu. Select any command from this new menu to get help for the real pulldown menu entry.

1−3: Requirements

Electric is written in the Java programming language and is distributed as a single ".jar" file, typically called "electric.jar". There are two variations on the ".jar" file: with or without source code. Either of these files can run Electric, but the one with source−code is larger because it also has all of the Java code.

Electric requires Java version 1.3 or later. It has been tested with Java version 1.5.

If you extract the source code from the ".jar" file and wish to build Electric, note that there are some Macintosh OS/X issues to consider.

- **Build on a Macintosh** The easiest thing to do is to remove references to "AppleJavaExtensions.jar" from the Ant script (build.xml). This package is a collection of "stubs" to replace Macintosh functions that are unavailable elsewhere. You can also build a native "App" by running the "mac−app" Ant script. This script makes use of files in the "packaging" folder.
- **Build on non−Macintosh, for non−Macintosh** If you are building Electric on and for a non−Macintosh platform, remove references to "AppleJavaExtensions.jar" from the Ant script (build.xml). Also, remove the module "com.sun.electric.MacOSXInterface.java". It is sufficient to delete this module, because Electric automatically detects its presence and is able to run without it.
- **Build on non−Macintosh, for all platforms** To build Electric so that it can run on all platforms, Macintosh and other, you will need to keep the module "com.sun.electric.MacOSXInterface.java". However, in order to build it, you will need the stub package "AppleJavaExtensions.jar". The package can be downloaded from Apple at http://developer.apple.com/samplecode/AppleJavaExtensions/AppleJavaExtensions.html.

Memory Control

One problem with Java is that the Java Virtual Machine has a memory limit. This limit prevents programs from growing too large, and speeds up garbage collection. However, it prevents large circuits from being edited. If Electric runs out of memory, you can request that more be used. To do this, use "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab). At the bottom of the dialog is a memory limit field. You will have to quit Electric and restart it for the new memory limit to take effect.

1−4: Setup

Running Electric varies with the different platforms:

- **UNIX/Linux** Although you may be able to double−click on the icon from some file managers, you can always run Electric from a shell window by typing:

```
java −jar electric.jar
```

- **Windows** You can run Electric by double−clicking on the file "electric.jar". You can also run Electric from a command window by typing:

```
java −jar electric.jar
```

- **Macintosh** Macintosh computers must be running OS 10.3 or later. You can run Electric by double−clicking on the file "electric.jar". You can also run Electric from a shell window by typing:

```
java −jar electric.jar
```

If the above form of the command does not work, try this alternate form:

```
java −classpath electric.jar com.sun.electric.Launcher
```

There are a number of options that can be given at the end of the command line:

- `−mdi` force a multiple document interface style (where Electric is one big window with smaller edit windows in it).
- `−sdi` force a single document interface style (where each Electric window is separate).
- `−s <script>` run the <script> file through the Bean shell.
- `−batch` run in batch mode (no windows or other user iterface are shown).
- `−version` provides full version information including build date.
- `−v` provides brief version information.
- `−help` prints a list of available command options.

1−5: Plug−Ins

Electric plug−ins are additional pieces of code that can be downloaded separately to enhance the system's functionality. Currently, these plug−ins are available:

- **IRSIM** The IRSIM simulator is a gate−level simulator from Stanford University. Although originally written in C, it was translated to Java so that it could plug into Electric. The Electric version is available from Static Free Software at www.staticfreesoft.com/electricIRSIM−8.02.jar.
- **Bean Shell** The Bean Shell is used to do parameter evaluation in Electric. Advanced operations that make use of cell parameters will need this plug−in. The Bean Shell is available from www.beanshell.org.
- **3D** The 3D facility lets you view an integrated circuit in three−dimensions. Although most of the viewer is included with Electric, there is one extra part that can be installed (a 3D axis controller). Also, none of the 3D display can function unless you have installed the Java3D package. The Java3D package is available from the Java Community Site, www.j3d.org. The 3D axis controller is available from Static Free Software at www.staticfreesoft.com/electricJava3D−8.02.jar
- **Animation** To enable 3D animation, you need the 3D facility (described above) as well as the Java Media Framework (JMF) and the animation code. The Java Media Framework is available from Sun Microsystems at java.sun.com/products/java−media/jmf and the animation code is available from Static Free Software at www.staticfreesoft.com/electricJMF−8.02.jar.

To attach a plugin, it must be invoked from the command line by adding it to the classpath. For example, to add the beanshell (a file named "bsh−2.0b1.jar"), type:

```
java -classpath electric.jar:bsh-2.0b1.jar com.sun.electric.Launcher
```

On Windows, you must use the ";" to separate jar files, and you might also have to quote the collection since ";" separates commands:

```
java -classpath "electric.jar;bsh-2.0b1.jar" com.sun.electric.Launcher
```

Note that you must explicitly mention the main Electric class (com.sun.electric.Launcher) when using plug−ins since all of the jar files are grouped together as the "classpath".

Chapter 1: Introduction

1−6: Fundamental Concepts

MOST CAD SYSTEMS use two methods to do circuit design: *connectivity* and *geometry*.

- The **connectivity** approach is used by every Schematic design system: you place components and draw connecting wires. The components remain connected, even when they move.
- The **geometry** approach is used by most Integrated Circuit (IC) layout systems: rectangles of "paint" are laid down on different layers to form the masks for chip fabrication.

ELECTRIC IS DIFFERENT because it uses connectivity for all design, even IC layout. This means that you place components (MOS transistors, contacts, etc.) and draw wires (metal−2, polysilicon, etc.) to connect them. The screen shows the true geometry, but it knows the connectivity too.

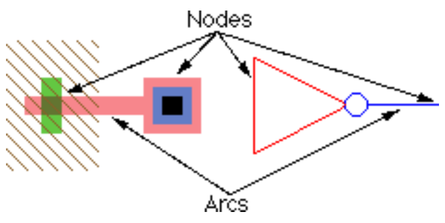
The advantages of connectivity−based IC layout are many:

- **No node extraction.** Node extraction is not a separate, error−prone step. Instead, the connectivity is part of the layout description and is instantly available. This speeds up all network−oriented operations, including simulation, layout−versus−schematic (LVS), and electrical rules checkers.
- **No geometry errors.** Complex components are no longer composed of unrelated pieces of geometry that can be moved independently. In paint systems, you can accidentally move the gate geometry away from a transistor, thus deleting the transistor. In Electric, the transistor is a single component, and cannot be accidentally destroyed.
- **More powerful editing.** Browsing the circuit is more powerful because the editor can show the entire network whenever part of it is selected. Also, Electric combines the connectivity with a layout constraint system to give the editor powerful manipulation tools. These tools keep the design well−connected, even as the circuit is modified on different levels of hierarchy.
- **Tools are smarter** when they can use connectivity information. For example, the Design Rule checker knows when the layout is connected and uses different spacing rules.
- **Simpler design process.** When doing schematics and layout at the same time, getting a correct LVS typically involves many steps of design rule cleaning. This is because node extraction must be done to obtain the connectivity of the IC layout, and node extractors cannot work when the design rules are bad. So, each time LVS problems are found, the layout must be fixed and made DRC clean again. Since Electric can extract connectivity for LVS without having perfect design rules, the first step is to get the layout and schematics to match. Then the design rules can be cleaned−up without fear of losing the LVS match.
- **Common user interface.** One CAD system, with a single user interface, can be used to do both IC layout and schematics. Electric tightly integrates the process of drawing separate schematics and has an LVS tool to compare them.

The disadvantages of connectivity−based IC layout are also known:

- **It is different** from all the rest and requires retraining. This is true, but many have converted and found it worthwhile. Users who are familiar with paint−based IC layout systems typically have a harder time learning Electric than those with no previous IC design experience.
- **Requires extra work** on the user's part to enter the connectivity as well as the geometry. While this may be true in the initial phases of design, it is not true overall. This is because the use of connectivity, early in the design, helps the system to find problems later on. In addition, Electric has many power tools for automatically handling connectivity.
- **Design is not WYSIWYG** (what−you−see−is−what−you−get) because objects that touch on the screen may or may not be truly connected. Electric has many tools to ensure that the connectivity has been properly constructed.

The way that Electric handles all types of circuit design is by viewing it as a collection of *nodes* and *arcs*, woven into a network. The nodes are electrical components such as transistors, contacts, and logic gates. Arcs are simply wires that connect two components. *Ports* are the connection sites on nodes where the wires connect.



In the above example, the transistor node has three pieces of geometry on different layers: polysilicon, active, and well. This node can be scaled, rotated, and otherwise manipulated without concern for specific layer sizes. This is because rules for drawing the node have been coded in a *technology* , which describes nodes and arcs in terms of specific layers.

Because Electric uses nodes and arcs for design, it is important that they be used to make all of the relevant connections. Although layout may appear to be connected when two components touch, a wire must still be used to indicate the connectivity to Electric. This requires a bit more effort when designing a circuit, but that effort is paid back in the many ways that Electric understands your circuit.

Besides creating meaningful electrical networks, arcs which form wires in Electric can also hold *constraints*. A constraint helps to control geometric changes, for example, the *rigid* constraint holds two components in a fixed configuration while the rest of the circuit stretches. These constraints propagate through the circuit, even across hierarchical levels of design, so that very complex circuits can be intelligently manipulated.

A *cell* is a collection of these nodes and arcs, forming a circuit description. There can be different *views* of a cell, such as the schematic, layout, icon, etc. Also, each view of a cell can have different *versions*, forming a history of design. Multiple views and versions of a cell are organized into *Cell groups*.

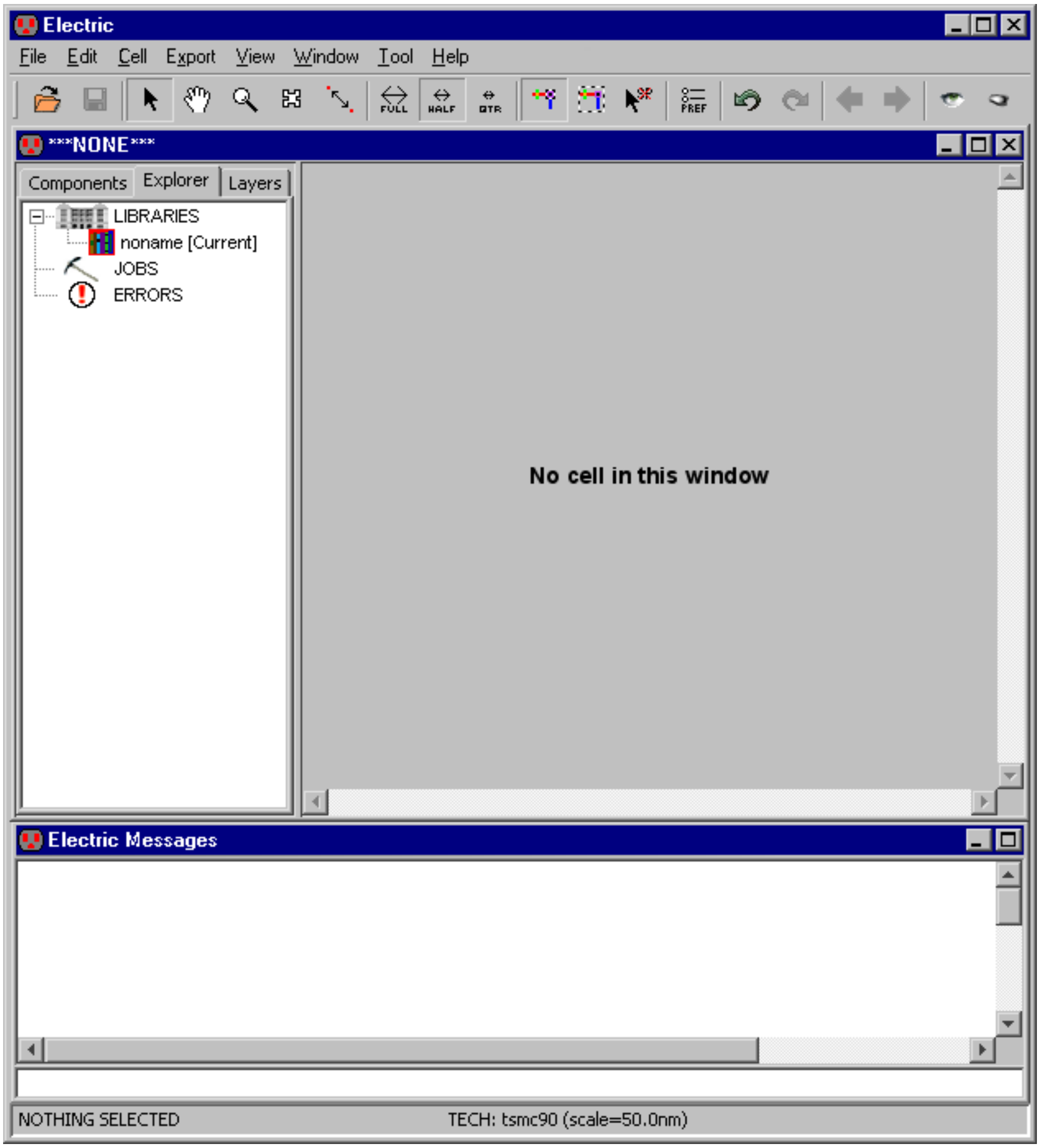
For example, a clock cell may consist of a schematic view and a layout view. The schematic view may have two versions: 1 (older) and 2 (newer). In such a situation, the clock cell group contains 3 cells: the layout view called "clock{lay}", the current schematic view called "clock{sch}", and the older schematic view called "clock;1{sch}".

Hierarchy is implemented by placing instances of one cell into another. When this is done, the cell that is placed is considered to be lower in the hierarchy, and the cell where it is placed is higher. Therefore, the notion of going *down* the hierarchy implies moving into a cell instance, and the notion of going *up* the hierarchy implies popping out to where the cell is placed. Note that cell instances are actually nodes, just like the primitive transistors and gates. By defining *exports* inside of a cell, these become the connection sites, or ports, on instances of that cell.

A collection of cells forms a *library*, and is treated on disk as a single file. Because the entire library is handled as a single entity, it can contain a complete hierarchy of cells. Any cell in the library can contain instances of other cells. A complete circuit can be stored in a single library, or it can be broken up into multiple libraries.

1–7: The Display

The Electric display varies from platform to platform. The image below shows a typical display with some essential features.



No cell in this window

NOTHING SELECTED TECH: tsmc90 (scale=50.0nm)

The *editing window* is the largest window that initially says "No cell in this window" (this indicates that no circuit is being displayed in that window). You can create multiple editing windows to see different parts of the design.

The left side of the edit window is the *side bar* that has 3 tabbed sections, the *components menu*, the *cell explorer*, and the *layers*.

The cell explorer lets you examine the hierarchy, system activity, and error messages (see Section 4−8 for more).

The *components menu* shows a list of nodes (blue border) and arcs (red border) that can be used in design. The arrangement of the entries in the components menu varies with the different technologies. For MOS technologies, see Section 7−4−2, for schematics, see Section 7−5−1, and for artwork, see Section 7−6−1.

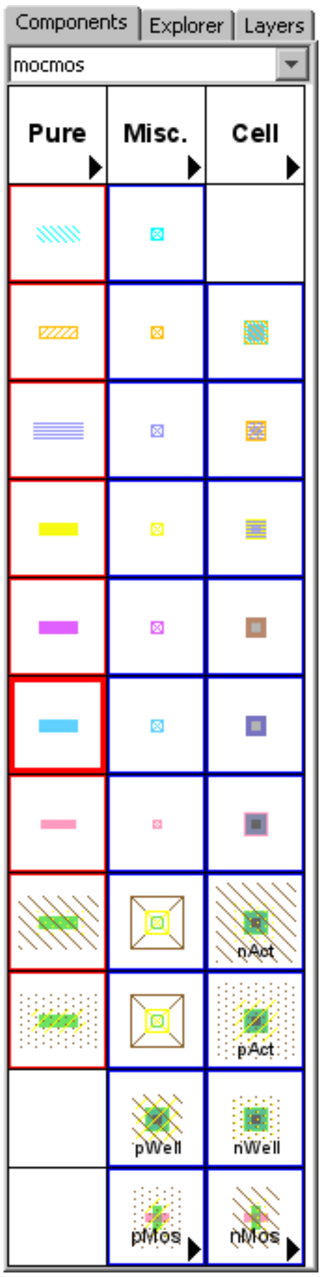
The top three entries in the components menu let you place pure−layer nodes (see Section 6−10−1), miscellaneous objects (see Section 7−6−3) and instances of cells (see Section 3−3).

The layers tab lets you control which parts of the display are visible. See Section 4−5 for more on layer visibility.

Below the edit window is the *messages window*, which is used for all textual communication.

Above the edit windows is a *pulldown menu* along the top with command options. On some operating systems, the pulldown menu is part of the edit window, and on others it is separate. Below the pulldown menu is a *tool bar* which has buttons for common functions.

Finally, the *status area* gives useful information about the design state. It appears along the bottom of the editing window or (in this example) at the bottom of the screen. The status area shows cursor coordinates, and can show global coordinates when traversing the hierarchy. To control the display of global coordinate values, use the "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab), and change the "Show hierarchical cursor coordinates in status bar" checkbox.



1−8: The Mouse

Electric makes use of only two mouse buttons: left and right. On systems with three−button mice, the middle button is not used. On Macintosh systems with only one button, the right button is obtained by holding the Command key when clicking.

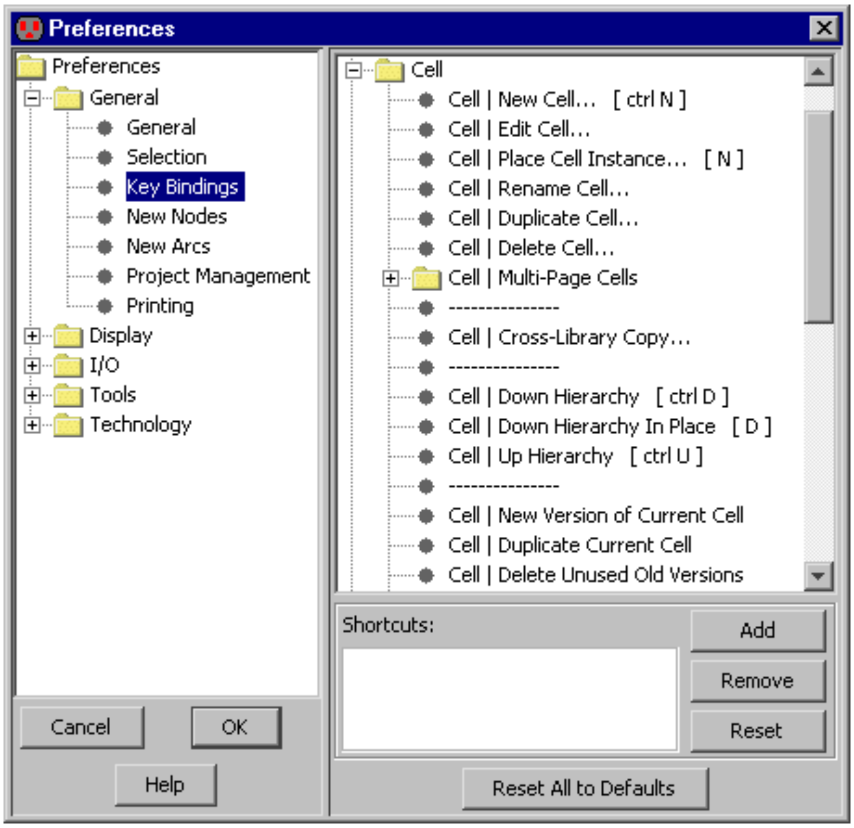
Modifier	Button	Action
	Left Mouse Click	Select
	Left Mouse Drag	Move selected objects
	Left Mouse Double Click	Get Info
CTRL	Left Mouse Click	Cycle through selected objects
SHIFT	Left Mouse Click	Invert selection
CTRL + SHIFT	Left Mouse Click	Cycle through objects to Invert
	Right Mouse Click	Draw or Connect Wire
SHIFT	Right Mouse Click	Zoom Out
SHIFT	Right Mouse Drag	Zoom In
CTRL + SHIFT	Right Mouse Drag	Draw Box
	Wheel Mouse Up/Down	Scroll Up/Down
SHIFT	Wheel Mouse Up/Down	Scroll Right/Left

By combining special keystrokes with the mouse functions, advanced layout operations can be done:

- **Switch Wiring Targets** Hit *Space* while holding the Right mouse button to switch between possible wiring targets under the mouse.
- **Switch Layers** Hit a number between *1−6* to switch layout layers. Additionally, if you have a port highlighted that can connect to the new layer, a contact cut will be created at that point and connected to the port.
- **Abort** Type *ESCAPE* to abort the current operation.

1–9: The Keyboard

Many common commands can be invoked by typing "quick keys" for them. These quick keys are shown in the pulldown menus next to the item. For example, the **New Cell...** command (in menu **Cell**) has the quick key "Control−N". On the Macintosh, the menu shows "⌘N", indicating that you must hold the command key while typing the "N"; on Windows and UNIX systems, the menu shows "Ctrl−N", indicating that you must hold the Control key while typing "N". There are also unshifted quick keys (for example, the letter "n" runs the **Place Cell Instance** command).



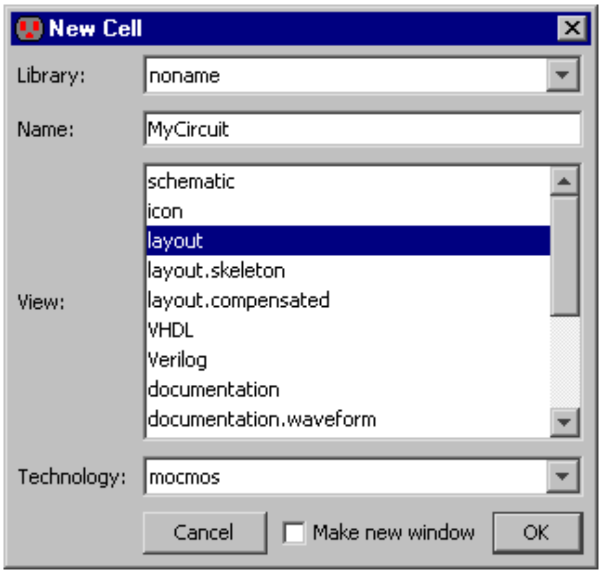
To change the bindings of quick keys, use the "Key Bindings" preferences (in menu **File / Preferences...**, "Tools" section, "Key Bindings" tab). The dialog shows the hierarchical structure of the pulldown menus on the top, and lets you add or remove key bindings in the bottom area.

You can remove a quick key binding with the "Remove" button, and you can add a quick key binding with the "Add" button. The "Reset" button restores default quick key bindings. Change key bindings with caution, because it customizes your user interface, making it more difficult for other users to work at your station.

You can get to EVERY menu command with key sequences. The keys to use are underlined in the menus. For example, the **File** menu has the "F" underlined, and the **Print...** command of that menu has the "P" underlined. This means that you can hold the Alt key and type "FP" to issue the print command.

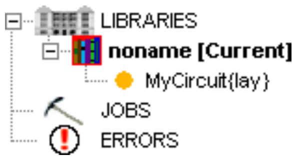
1−10−1: IC Layout Example: Make a Cell

This section takes you through the design of some simple IC layout.



Before you can place any IC layout, the editing window must have a cell in it. Use the **New Cell...** command (in menu **Cell**). This will show a dialog that lets you type a new cell name. Type the name ("MyCircuit" is used here) and click OK. The editing window will no longer have the "No cell in this window" message, and circuitry may now be created.

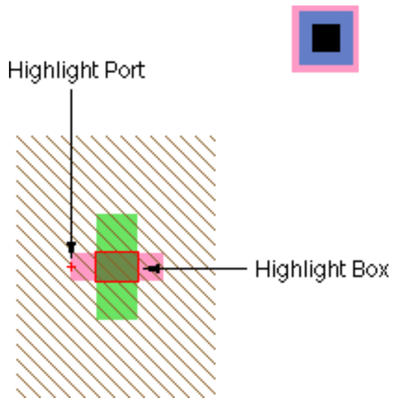
After creating a cell, look at the cell explorer (in the status bar on the left side of the edit window). Under the "LIBRARIES" icon, you will see the list of libraries (currently only one called "noname"). If you open that library's icon, you will see the cells in the library (currently only "MyCircuit").



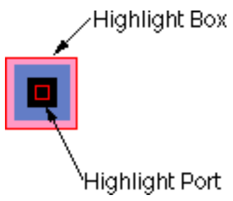
1−10−2: IC Layout Example: Create a Node

Layout is placed by selecting nodes from the side bar's components menu, and then wiring them together. This example shows two nodes that have been created. This was done by clicking on the appropriate component menu entry, and then clicking again in the editing window to place that node. After clicking on the component menu entry, the cursor changes to a pointing hand to indicate that you must select a location for the node. When placing the node, if you press the button and do not release it, you will see an outline of the new node, which you can drag to its proper location before releasing the button.

In this example, the top node is called Metal−1−Polysilicon−1−Con (a contact between metal layer 1 and polysilicon layer 1, found in the fifth entry from the bottom in the right column of the component menu). The node on the bottom is called N−Transistor (lower−right entry of the component menu). Both of these nodes are from the MOSIS CMOS technology (which is listed as "mocmos" in the status area).



1−10−3: IC Layout Example: Highlighting



A *highlighted* node has two selected areas: the node and a port on that node. Note that the transistor is highlighted in the previous example, and the contact is highlighted in the example here. The larger selected area covers the node, and it surrounds the "important" part (for example, on the Transistor, it covers only the overlap area, excluding the tabs of active and gate on the four sides). The smaller selected area is the currently highlighted port (there are four possible ports on the transistor, but only one on the contact).

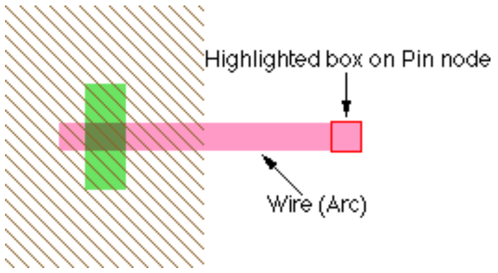
To highlight a node, use the *left* button. The node, and the closest port to the cursor, will be selected. After highlighting, you can hold the mouse button down and drag the highlighted object to a new location. If nothing is under the cursor when the selection button is pushed, you may drag the cursor while the button remains down to define an area in which all objects will be selected.

Another way to affect what is highlighted is to use the *shift−left* button. This button causes object highlighting to be reversed (highlighted objects become unhighlighted and unhighlighted objects are highlighted).

The shape of the highlighted port is important. Ports are the sites of arc connections, so the end point of the arc must fall inside this port area. Ports may be rectangles, lines, single points (displayed as a "+"), or any arbitrary shape. For example, when the active tabs of a transistor are highlighted, the port is shown as a line.

1−10−4: IC Layout Example: Make an Arc

To wire a component, select it, move the cursor away from the component, and use the *right* button. A wire will be created that runs from the component to the location of the cursor. Note that the wire is a fixed−angle wire which means that it will be drawn along a horizontal or vertical path from the originating node.



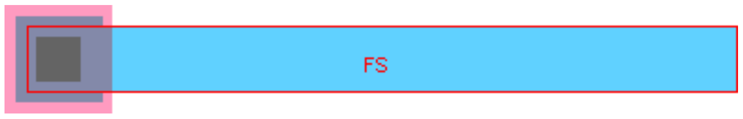
To see where the wire will end, click but do not release the button and drag the outline of the wire's terminating node (a pin) until it is in the proper location. It is highly recommended that you do all wiring operations this way, because wiring is quite complex and can follow many different paths.

Once a wire has been created, the other end is highlighted (see above). This is the highlighting of a *pin* node that was created to hold the other end of the arc. Because it is a node, the *right* button can be used again to continue the wire to a new location. If, during wiring, the cursor is dragged on top of an existing component, the wire will attach to that component.

To remove wires or components, you can issue the **Undo** command (in menu **Edit**) to remove the last created object. Alternatively, you can select the component and use the **Erase** command (in menu **Edit**).

1−10−5: IC Layout Example: Constraints

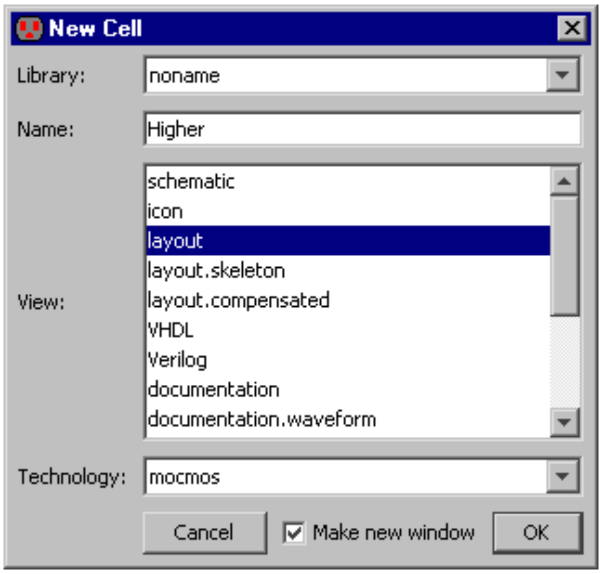
Once components are wired, moving them will also move their connecting wires. Notice that the wires stretch and move to maintain the connections. What actually happens is that the programmable constraint system follows instructions stored on the wires, and reacts to node changes. The default wire is *fixed−angle* and *slidable*, so the letters "FS" are shown when the wire is highlighted.



Select a wire and issue the **Rigid** command (in menu **Edit / Arc**). The letters change to "R" on the arc and the wire no longer stretches when nodes move. Find another arc and issue the **Not Fixed−angle** command. Now observe the effects of an unconstrained arc as its neighboring nodes move. These arc constraints can be reversed with the **Rigid** and **Fixed−angle** commands. See Section 5−2−1 for more on these constraints.

1−10−6: IC Layout Example: Hierarchy

Electric supports hierarchy by allowing you to place instances of another cell. These instances are nodes, just like the simpler ones in the component menu. To see hierarchy in action, create a new cell with the **New Cell...** command (in menu **Cell**). Make sure the "Make new window" option is checked in the dialog. Then type the new cell name ("Higher" is used in the example here).



New Cell

Library: noname

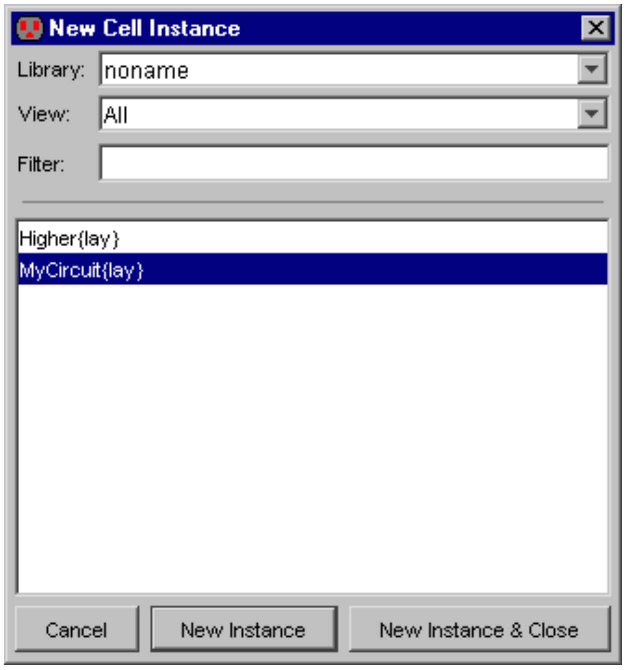
Name: Higher

View: schematic, icon, layout (selected), layout.skeleton, layout.compensated, VHDL, Verilog, documentation, documentation.waveform

Technology: mocmos

[✓] Make new window

A new (empty) cell will appear in a separate window. Try creating a few simple nodes in this new window (place a contact or two).



New Cell Instance

Library: noname

View: All

Filter:

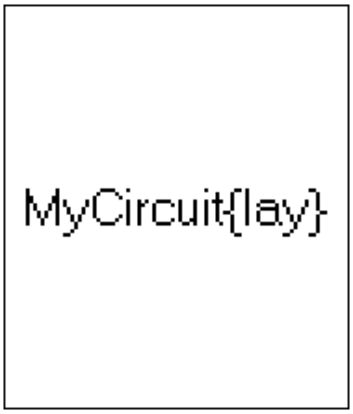
Higher{lay}

MyCircuit{lay} (selected)

Now place an instance of the other cell by using the **Place Cell Instance...** command (in menu **Cell**). You can also click the "Cell" entry in the component menu. You will be given a list of cells to create: select the one that is in the OTHER window (the one called "MyCircuit" in this example). Then click in the newer cell to create the instance.

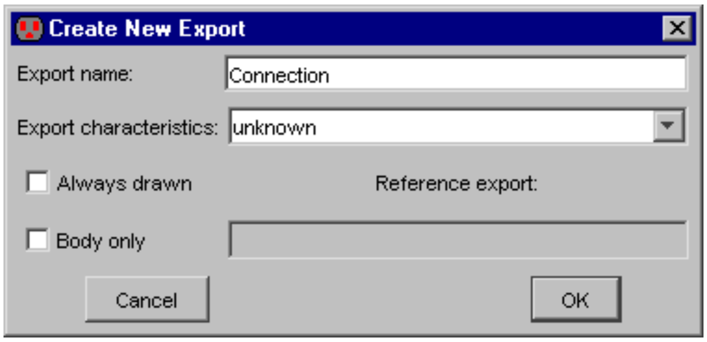
The box that appears is a node in the same sense

as the contacts and transistors: it can be moved, wired, and so on. In addition, because the node contains subcomponents, you can see its contents by selecting it and using the **One Level Down** command (in menu **Cell / Expand Cell Instances**, or click on the opened−eye button in the tool bar). Note that if the objects in a cell no longer fit in the display window, use the **Fill Display** command (in menu **Window**).



1−10−7: IC Layout Example: Exports

Before you can attach wires to the instance node, there must be connection sites, or *ports* on that node. Primitive nodes such as contacts and transistors already have their ports established, but you must explicitly create ports for cell instances.


Create New Export

Export name: Connection

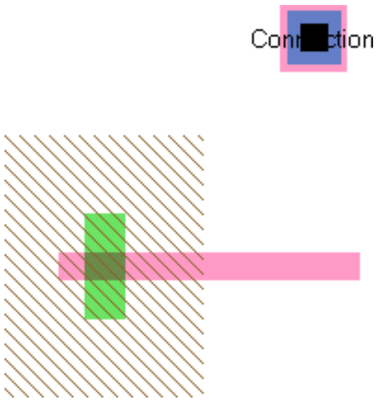
Export characteristics: unknown

☐ Always drawn

Reference export:

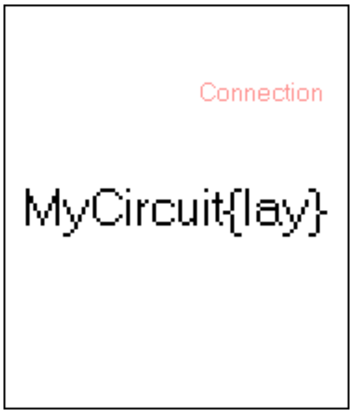
☐ Body only

This is done by creating *exports* inside the cell definition. Move the cursor to the window with the lower−level cell ("MyCircuit") and select the contact node. Then issue the **Create Export...** command (in menu **Export**). You will be prompted for an export name and its characteristic (the characteristics can be ignored for now).



This takes the port on the contact node and exports it to the outside world. Its name will be visible on the unexpanded instance node in the higher−level cell.

You can now connect wires to that node in just the same way as you wired the contact.

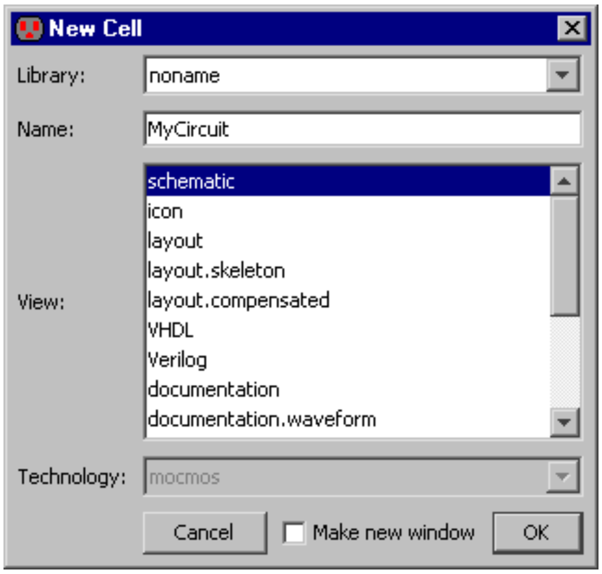


1−10−8: IC Layout Example: Final Points

Some final commands that should be mentioned in this introductory example are the **Save Library** and the **Quit** commands which can be found in the **File** menu. They do the obvious things.

1−11−1: Schematics Example: Make a Cell

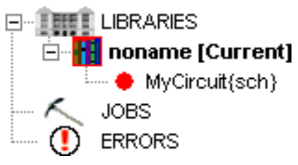
This section takes you through the design of some simple schematics.



Before you can place any schematics, the editing window must have a cell in it. Use the **New Cell...** command (in menu **Cell**). Type the name ("MyCircuit" is used here) and select the "schematic" view.

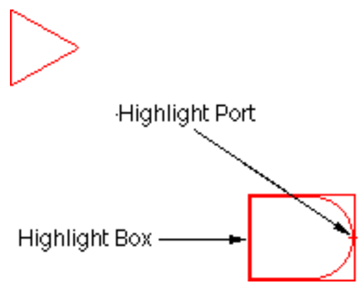
The editing window will no longer have the "No cell in this window" message, and circuitry may now be created. Note that the component menu on the left will change to show schematics primitives. Also, the Schematic technology is now listed in the status area at the bottom of the screen.

After creating a cell, look at the cell explorer (in the status bar on the left side of the edit window). In the "LIBRARIES" icon, you will see the list of libraries (currently only one called "noname"). If you open that library's icon, you will see the cells in the library (currently only "MyCircuit").



1−11−2: Schematics Example: Make a Node

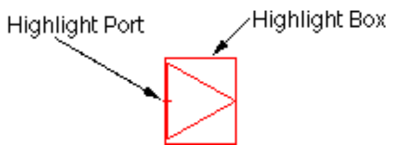
Schematic nodes are placed by selecting them from the side bar's components menu (on the left), and then wiring them together. This example shows two nodes that have been created. This was done by clicking on the appropriate component menu entry, and then clicking again in the editing window to place that node.



After clicking on the component menu entry, the cursor changes to a pointing hand to indicate that you must select a location for the node. When placing the node, if you press the button and do not release it, you will see an outline of the new node, which you can drag to its proper location before releasing the button.

In this example, the top node is called a Buffer (found on the right side of the component menu in the third entry from the top). The node on the bottom is called an And (top entry on the right).

1−11−3: Schematics Example: Highlighting



A *highlighted* node has two selected parts: the node and a port on that node. Note that the And is highlighted in the previous example, and the Buffer is highlighted in the example here. The little "+" sign is the currently highlighted port (there are two possible ports on these nodes, on the input and the output).

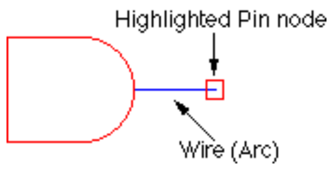
To highlight a node, use the *left* button. The node, and the closest port to the cursor, will be selected. After highlighting, you can hold the mouse button down and drag the highlighted object to a new location. If nothing is under the cursor when the selection button is pushed, you may drag the cursor while the button remains down to define an area in which all objects will be selected.

Another way to affect what is highlighted is to use the *shift−left* button. This button causes object highlighting to be reversed (highlighted objects become unhighlighted and unhighlighted objects are highlighted).

The shape of the highlighted port is important. Ports are the sites of arc connections, so the end point of the arc must fall inside this port area. Ports may be rectangles, lines, single points (displayed as a "+"), or any arbitrary shape. For example, the entire left side of the And gate is the input port and so its highlighting is a line.

1−11−4: Schematics Example: Make an Arc

To wire a component, select it, move the cursor away from the component, and use the *right* button. If you click the right button and hold it without releasing, then you can move around and see where the wire will go when you do release.



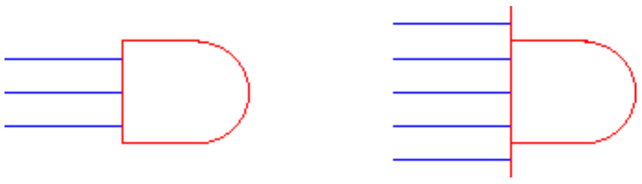
A wire will be created that runs from the component to the location of the cursor. Note that the wire is a fixed−angle wire which means that it will be drawn along a horizontal, vertical, or 45−degree path from the originating node. To see where the wire will end, click but do not release the button and drag the outline of the wire's terminating node (a pin) until it is in the proper location. It is highly recommended that you do all wiring operations this way, because wiring is quite complex and can follow many different paths.

Once a wire has been created, the other end is highlighted (see above). This is the highlighting of a *pin* node that was created to hold the other end of the arc. Because it is a node, the *right* button can be used again to continue the wire to a new location. If, while wiring, the dragged location is over an existing component, the wire will attach to that component.

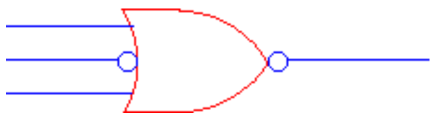
To remove wires or nodes, you can issue the **Undo** command (in menu **Edit**) to remove the last created object. Alternatively, you can select the component and use the **Erase** command (in menu **Edit**).

1−11−5: Schematics Example: Multi−Input gates and Negation

One aspect of the And, Or, and Xor gates that you will notice is that their left side (the input side) can accept any number of wires. To see this in action, place one of these components in the cell. Then repeatedly select its left side and use the *right* button to draw wires out of it. Each wire will connect at a different location in the input port, and once the side fills with arcs, it will automatically grow to fit more. Note that the vertical cursor location along the input side is used to select the position that will be used when a new wire is added.

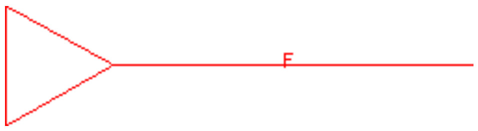


To negate an input or output of a digital gate, select the port or the arc and use the **Toggle Port Negation** command (in menu **Edit / Technology Specific**). With this facility, you can construct arbitrary gate configurations.



1−11−6: Schematics Example: Constraints

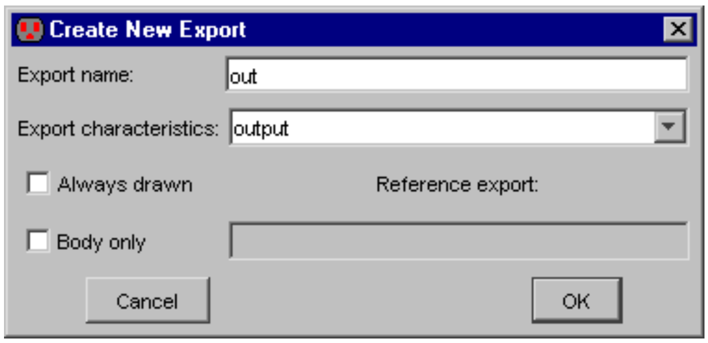
Once components are wired, moving them will also move their connecting wires. Notice that the wires stretch and move to maintain the connections. What actually happens is that the programmable constraint system follows instructions stored on the wires, and reacts to component changes. The default wire is *fixed−angle*, so the letter "F" is shown when the wire is highlighted.



Select a wire and issue the **Rigid** command (in menu **Edit / Arc**). The letter changes to "R" on the arc and the wire no longer stretches when components move. Find another arc and issue the **Not Fixed−angle** command. Now observe the effects of an unconstrained arc as its neighboring nodes move. These arc constraints can be reversed with the **Rigid** and **Fixed−angle** commands. See Section 5−2−1 for more on these constraints.

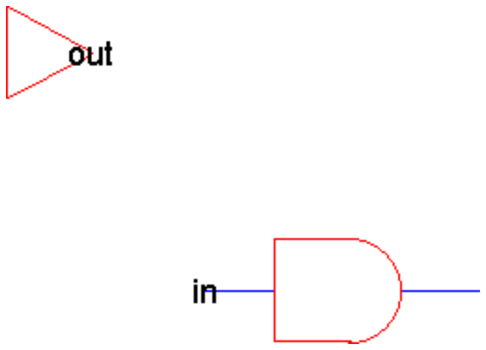
1−11−7: Schematics Example: Hierarchy and Icons

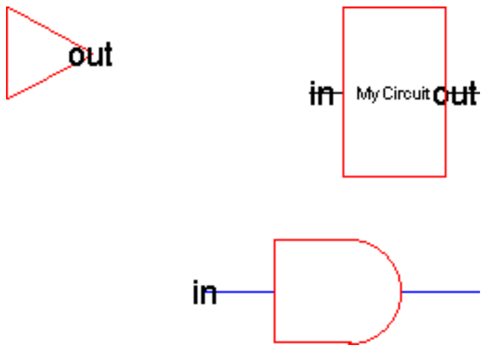
Electric supports hierarchy by allowing you to create icons for a schematic and place them in another cell. Before creating an icon, all connection points to the schematic should be defined.



To define connection points for a schematic, you must create *exports* on the schematic. To see an example of this, select the output port of the Buffer node and issue the **Create Export...** command (in menu **Export**). You will be prompted for an export name and its characteristics (set the characteristics to "output").

The output port on the buffer node is now exported to the outside world. Run a wire from the input side of the And node and export the pin at the end of the wire. Your circuit should look like this.

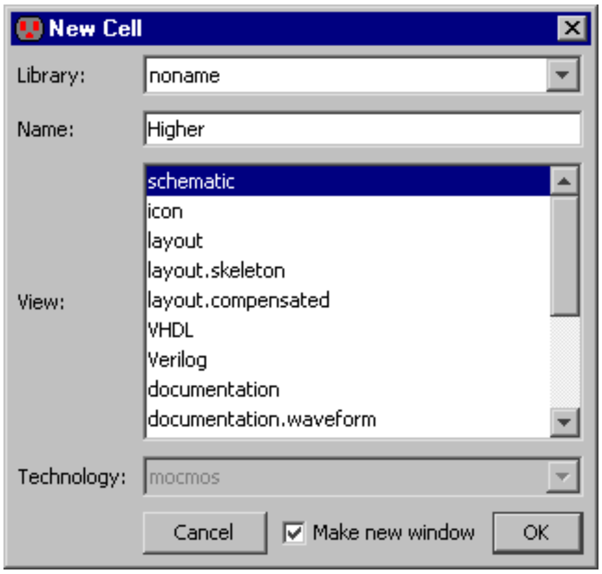




You can now make an icon for this circuit by using the **Make Icon** command (in menu **View**). The icon will be placed in your circuit (you may have to move it away from the rest of the circuitry). The result will look like this.

To test this icon in a circuit, create a new cell in which to place instances of the icon. Use the **New Cell...** command (in menu **Cell**). Type the new cell name ("Higher" is used in the example here) and make sure its view is "schematic".

A new (empty) cell will appear in a separate window. Try creating a few simple nodes in this new window (place a gate or two).

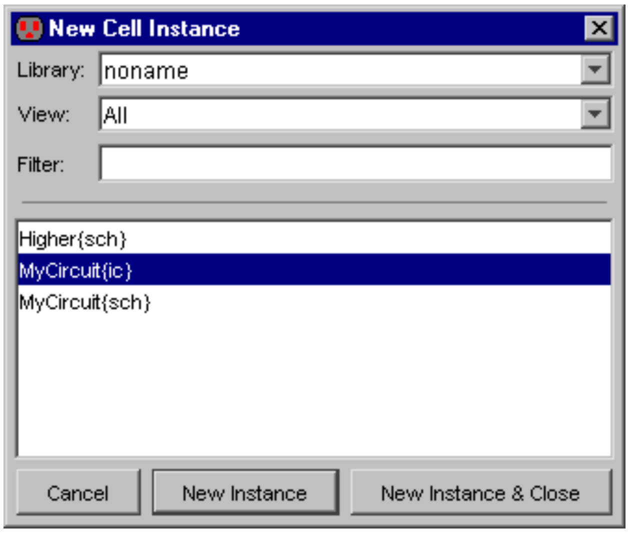

New Cell

Library: noname

Name: Higher

View: schematic

Technology: mocmos


New Cell Instance

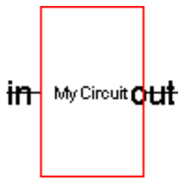
Library: noname

View: All

Filter:

Now place an instance of the other cell by using the **Place Cell Instance...** command (in menu **Cell**). You can also click the "Cell" entry in the component menu. You will be given a list of cells to create: select the one that is in the OTHER window (the one called "MyCircuit{ic}" in this example). Then click in the newer cell to create the instance.

The icon that appears is a node in the same sense as the Buffer and And gate: it can be moved, wired, and so on. In addition, because the node contains subcomponents, you can see its contents by selecting it and using the **Down Hierarchy** command (in menu **Cell**). Note that if the objects in a cell no longer fit in the display window, use the **Fill Window** command (in menu **Window**).



1−11−8: Schematics Example: Final Points

Some final commands that should be mentioned in this introductory example are the **Save Library** and the **Quit** commands which can be found in the **File** menu. They do the obvious things.

Chapter 2: Basic Editing

2−1−1: Selecting Nodes and Arcs

Electric is a *noun/verb* system, meaning that all commands work by first selecting something (the noun) and then doing an operation (the verb). For this reason, selection is important.

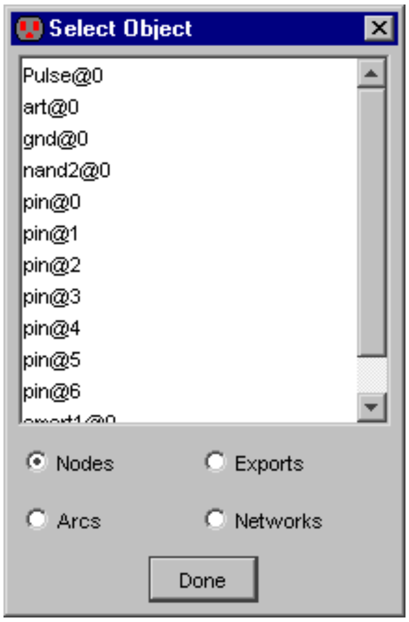
Selection (and movement, wiring, and zooming) are done in "selection" mode, which is the default mode. This mode is indicated by having the "selection" icon highlighted in the tool bar.

Selection is done with clicks of the *left* button. Individual nodes and arcs are selected by clicking over them. You can tell in advance what will be selected by the button click, because the next object to be selected is shown in blue. This advance selection is called "mouse−over highlighting" and can be disabled (see section 2−1−4). Once selected, objects are highlighted on the screen. If you use the *shift−left* button, unhighlighted nodes and arcs are added to the selection, but objects that are already highlighted become deselected.

There are often multiple objects under the cursor (for example, in the area where an arc overlaps a node). To get the object you want, hold the control key while clicking. The *control−left* button cycles through all objects under the cursor.

The notion of toggling selection (shift−left) and cycling through what is under the cursor (control−left) can be combined. If there are multiple objects under the cursor, and you are trying to toggle the selection, use the *control−shift−left* button to cycle through them .

To select an object by its name, use the **Select Object...** command (in menu **Edit / Selection**). The resulting dialog lets you select nodes, arcs, exports, or networks in the cell.


Select Object

Pulse@0
art@0
gnd@0
nand2@0
pin@0
pin@1
pin@2
pin@3
pin@4
pin@5
pin@6

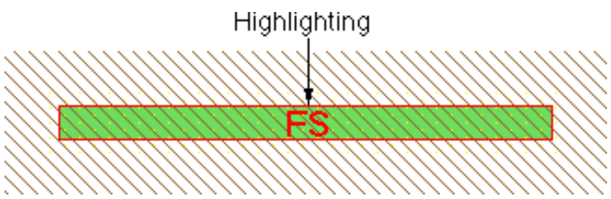
[✓ Nodes] Exports
Arcs Networks

Done

To select everything in the cell, use the **Select All** command (in menu **Edit / Selection**). To deselect everything, use **Select Nothing**.

To select everything in the cell that is the same as the currently selected objects, use the **Select All Like This** command (in menu **Edit / Selection**). The **Deselect All Arcs** command deselects all selected arcs. This is useful when you wish to select a set of nodes, but you have selected the entire area, including nodes and arcs.

2−1−2: Selection Appearance

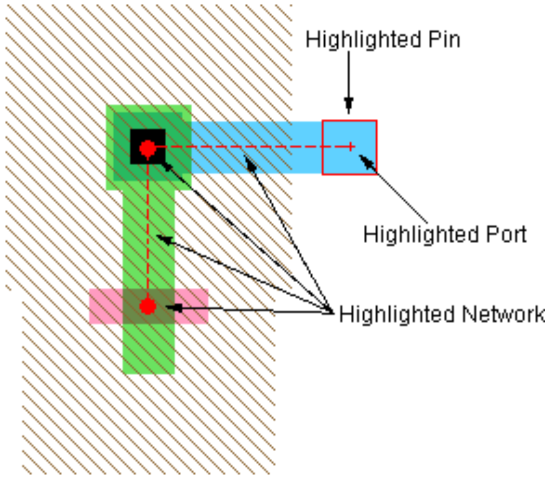


Highlighted objects have a box drawn around them. In some cases, the object extends beyond the box, but the box encloses the essential part of the object. For example, MOS transistors are highlighted where the two materials cross, even though the materials extend on all four sides. Also, CMOS active arcs have implants that surround them, but the highlight covers only the central active part.

Besides the basic box, there will be other things drawn when an object is highlighted. Highlighted arcs have their constraint characteristics displayed. The example above shows an arc that is both fixed−angle ("F") and slidable ("S"). The letter "R" is used for rigid arcs, and an "X" appears when none of these constraints apply. See Section 5−1 for more information on arc constraints.

When nodes are selected, a port is also highlighted. The port that is highlighted is the one closest to the cursor when the node is selected. If the port is a single point, you see a "+" at the port. If the port is larger than a single point, it is shown as a line or rectangle.

Highlighted nodes will also show the entire network that extends out of the highlighted port. Arcs in that network will be drawn with dashed lines, and nodes in that network will be indicated with dots. The example here shows the highlighting of a pin node (in the upper−right) with a single−point port ("+") which is connected to a contact and a transistor.



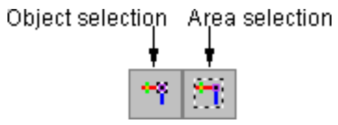
It is important to understand that Electric is not exactly a WYSIWYG editor (what−you−see−is−what−you−get). Nodes that are touching on the screen may not actually be connected if there are no arcs joining them. The best way to ensure that the circuit is correct is to highlight a node and see the extent of the connections on it.

2−1−3: Unusual Selection: Areas and Text

Besides highlighting nodes and arcs, Electric can also highlight an arbitrary rectangular area. The notion of a *highlighted area*, as opposed to a *highlighted object*, is used in some commands, and it generally implies highlighting of everything in the area.

There are two ways to highlight an area. If you click the *left* button where there is no object, and hold it down while dragging over objects, all of those objects will be highlighted.

To more precisely define a highlighted area, switch to area selection (as opposed to object selection) with the **Area** command (in menu **Edit / Modes / Select**, or click on the "Area Selection" icon in the tool bar).



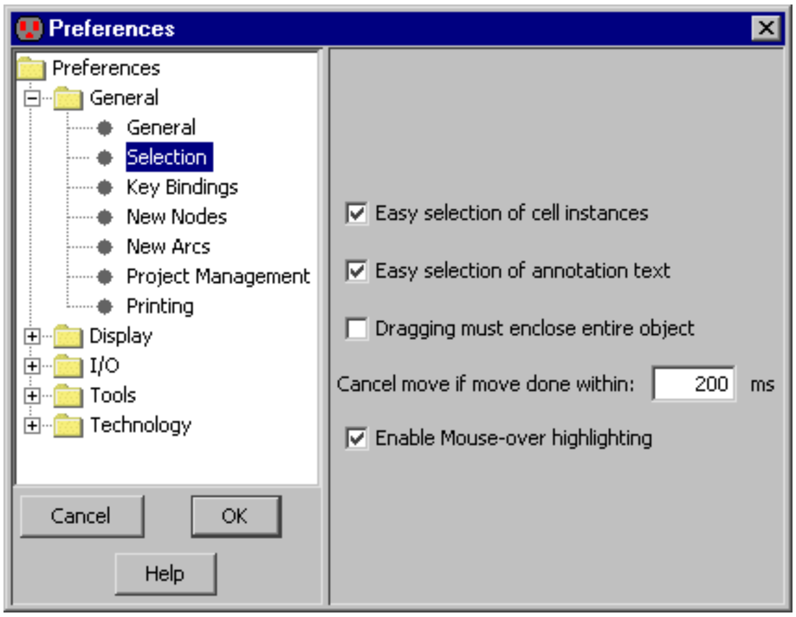
Once in area selection mode, each click and drag of the *left* button leaves the highlight rectangle on the screen exactly as it was drawn. You can convert this selection to a set of actual nodes and arcs with the **Enclosed Objects** command (in menu **Edit / Selection**).

Selecting Text

Highlighted text appears as an "X" over the letters. However, text is a special case, so it will not be covered until later (section 6−8−2). For now, if you highlight some text, it is best to click again and select something else.

2−1−4: Controlling Selection

Once a selection is made, you can save it with the **Push Selection** command (in menu **Edit / Selection**). The highlighting is not changed, but it is saved on a stack. To restore this selection at a later time, use the **Pop Selection** command.



☑ Easy selection of cell instances

☑ Easy selection of annotation text

☐ Dragging must enclose entire object

Cancel move if move done within: 200 ms

☑ Enable Mouse-over highlighting

There are some selection preferences that can be set with "Selection" preferences (in menu **File / Preferences...**, "General" section, "Selection" tab). "Easy selection" controls whether objects can be selected with simple clicks, or whether they require extra effort to select. You can request that all cell instances be hard to select and that all annotation text be hard to select. See the (next section) for more on this.

The "Dragging must enclose entire object" requests that area−selection completely enclose an object in order to select it. The default is that any object touching the area is selected.

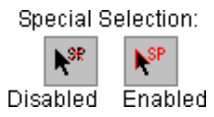
To prevent accidental moving of an object after selecting it, object movement is disabled for a short time after the selection click. This delay can be controlled.

When the cursor roams over a circuit, it shows a "preview" of what will be selected by the next click. The advance preview is shown in a different color than the actual highlighting (initially blue, but this can be changed with the "Color" Preferences, see Section 4−6−2). This feature is called "mouse−over highlighting". If you do not want to see this preview, uncheck "Enable Mouse−over highlighting".

2−1−5: Easy and Hard Selection

In a busy circuit, many objects may overlap, causing confusion when selecting. To simplify selection, objects can be marked so that they are no longer *easy−to−select*, which means that standard selection does not work on them.

To select hard−to−select objects, use the **Special Select** command (in menu **Edit / Modes / Select**). You can also click on the "Special Select" tool bar button to enable "special selection".



Ease of selection extends to more than just nodes and arcs. There are four "classes" of objects that can be selected:

- Basic objects (all arcs, primitive nodes, and port names)
- Cell instances
- Annotation text (names and other text placed on nodes and arcs)
- Instance text (an unexpanded cell instance's name)

By default, the first three classes are easy−to−select, and instance text is hard−to−select. If you uncheck "Easy selection of cell instances" in the selection preferences dialog, then cell instances become hard−to−select. If you uncheck "Easy selection of annotation text" in the selection preferences dialog, then annotation text becomes hard−to−select.

Although all nodes and arcs are typically easy−to−select, you can control them individually by unchecking the "Easy to Select" field in their properties dialog (use the **Object Properties...** command in menu **Edit / Properties**). If multiple objects are selected, the **Object Properties...** dialog has a popup in the lower−right for changing their selection difficulty.

Special commands exist in the **Selection** menu for dealing with easy−to−select nodes and arcs. You can select all of the easy−to−select objects in the current cell with the **Select All Easy** command. Similarly, you can select those that are not easy−to−select with the **Select All Hard** command. To change the ease of selection for a set of objects, highlight them and use either **Make Selected Easy** or **Make Selected Hard**.

2−2−1: Node Creation

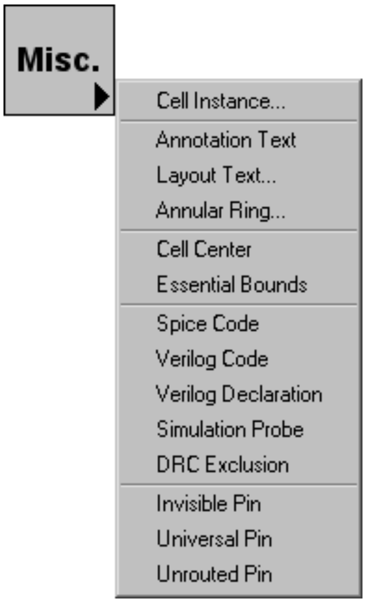
Node creation is done by selecting a node from the component menu in the side bar (on the left). The nodes are outlined in blue. After clicking on one of these nodes, click in the edit window to place the node.

The location of the cursor is aligned to the nearest grid unit. This adjustment can be controlled with the "Grid" preferences (in menu **File / Preferences...**, "Display" section, "Grid" tab).

When placing a node, the cursor points to the *anchor point* of the newly created node. This is the center (for primitives) or the location of the cell−center (for cell instances). Cell instances can change their anchor point by moving the Cell−Center node inside of their layout (see Section 3−3).

Besides basic components, there are special entries in the component menu for creation of additional nodes:

- The "Cell" button displays a list of cell instances that can be created (see Section 3−3).
- The "Pure" button (only available in layout technologies) lets you place pure−layer nodes (see Section 6−10−1).
- The "Spice" button (only available in schematics) lets you place Spice primitives (see Section 9−4−3).
- The "Misc" button has a collection of special objects that can be created.



These objects can be created with the "Misc" button:

- **Cell Instance** brings up a dialog to select an instance to place (see Section 3−3).
- **Annotation Text** places a node that contains only text (see Section 6−8−1).
- **Layout Text...** brings up a dialog to create text from layout nodes (see Section 6−10−3).
- **Annular Ring...** brings up a dialog to create circular shapes (see Section 6−10−3).
- **Cell Center** places a node that defines the origin of the cell (see Section 3−3).
- **Essential Bounds** places a node that defines the corners of the cell's essential bounds (see Section 7−6−3).
- **Spice Code** places a text−only node that will be inserted into Spice decks (see Section 9−4−3).
- **Verilog Code** places a text−only node that will be inserted into the code area of Verilog decks (see Section 9−4−2).
- **Verilog Declaration** places a text−only node that will be inserted into the declaration area of Verilog decks (see Section 9−4−2).
- **Simulation Probe** places a node that can be used to display simulation results (see Section 4−12−1).

- **DRC Exclusion** places a node that covers DRC errors and causes them to be ignored (see Section 9−2−5).
- **Invisible Pin** places an invisible−pin node (see Section 7−6−3).
- **Universal Pin** places an universal−pin node (see Section 7−6−3).
- **Unrouted Pin** places an unrouted−pin node (see Section 7−6−3).

2−2−2: Arc Creation

As the introductory example showed, arcs are created by clicking the *right* button. This can actually function in two different ways, depending on what is highlighted.

If one node is highlighted, *segment wiring* is done, in which an arc is drawn from the highlighted node to the location of the cursor. If there is nothing at that location, a pin is created, and it is left highlighted. Using the *right* button again runs an arc from the pin to another location. By clicking and holding the *right* button, you can see the path that the new arc will follow.

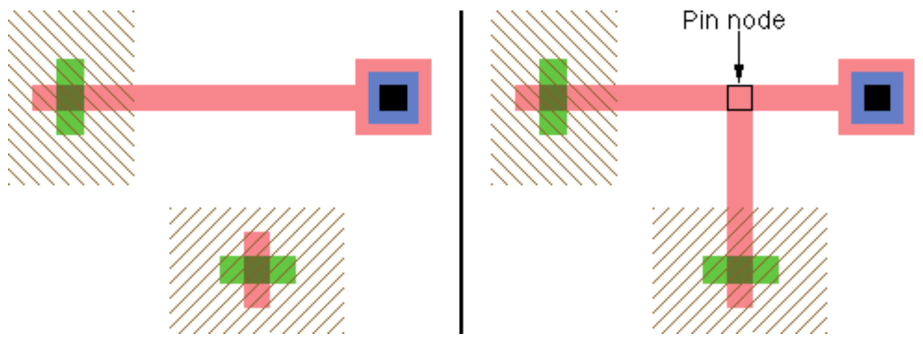
In general, all wiring operations should be done by clicking and holding the *right* button, then moving the cursor until the intended wiring is shown, and finally releasing. This is recommended because wiring is quite complex and can follow many different paths.

If you type a digit key while the right button is pressed, it changes the wiring layer by inserting contacts to that layer of metal. For example, if you are running a metal−1 wire, and type "3" during the wiring, then two contacts will be added (metal−1−metal−2 and metal−2−metal−3) to make the wire run in metal−3.

If the cursor is over another object when the *right* button is released, the new wire attaches to that object. If there are multiple objects under the cursor, press the *space bar* (while the right button is pressed) to cycle through the possible endpoints (including the possibility of connecting to none of them).

The other way that the creation button can operate is *two−point wiring*, in which two nodes are highlighted and one or more arcs are created to connect them. Highlighting of these two nodes is done by clicking the *left* button over the first one, and then using the *shift−left* button on the second. Note that if the second node is obscured by other objects, you can cycle through the objects under the cursor with the *control−shift−left* button. Once the two nodes are highlighted, use the *right* button to wire them together. Note that the highlighted ports on the selected nodes are important: arcs will run between them, so they must be compatible in their wiring capabilities.

Two−point wire creation first attempts to run a single arc. Generally, this can happen only if the ports are lined up accurately. Failing single arc placement, an attempt is made to connect with two arcs and an intermediate node. These two arcs can bend in one of two directions, determined by the location of the cursor.



In addition to running an arc between two nodes, you can also use arcs as the starting or ending point of arc creation. If it is sensible, the creation command actually uses one of the nodes on an end of the selected arc.

However, if the connection falls inside the arc, it is split and a new node is created to make a "T" connection.

Electric will allow you to connect two nodes or arcs as long as there is some way in the current technology for those objects to be connected. For example, if connecting between metal−1−pin and a metal−3−pin in the MOSIS CMOS technology, Electric will place metal−1−metal−2 and metal−2−metal−3 contact cuts down, and wire between all four nodes. When vias are inserted, they are placed closest to the "destination" node (or farthest from the original node).

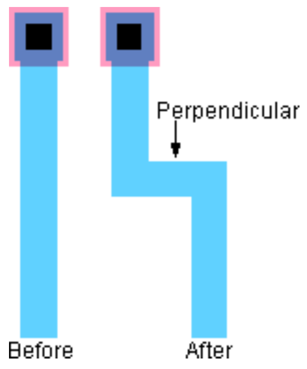
As mentioned in Section 1−8, pressing the number keys for a valid layer switches to that layer. If a node is highlighted, it will route to that layer from the node, creating contacts as necessary.

2−2−3: Special Cases

The default width is set by the "New Arcs" preferences (in menu **File / Preferences...**, "General" section, "New Arcs" tab). If there are other arcs of this type already connected to the new one, and they are wider than normal, then the new arc will use that width.

Note that all arcs overlap their endpoint by half of their width, so very wide arcs may overlap their destination with too much geometry. You can turn off this overlap by using the **Toggle End Extension of Head** and **Toggle End Extension of Tail** commands (in menu **Edit / Arc**). See Section 5−4−3 for more on end extension.

An unusual circuit creation command is the **Insert Jog In Arc** command (in menu **Edit / Arc**). This command inserts a jog in the highlighted arc by replacing it with three new arcs. Two of the new arcs run to the location of the cursor, and the third arc is perpendicular to them, connecting the ends at the cursor location (initially it has zero length). Once the jog is inserted, either half of the arc may be moved without affecting the other half, and the perpendicular arc will keep the circuit connected.



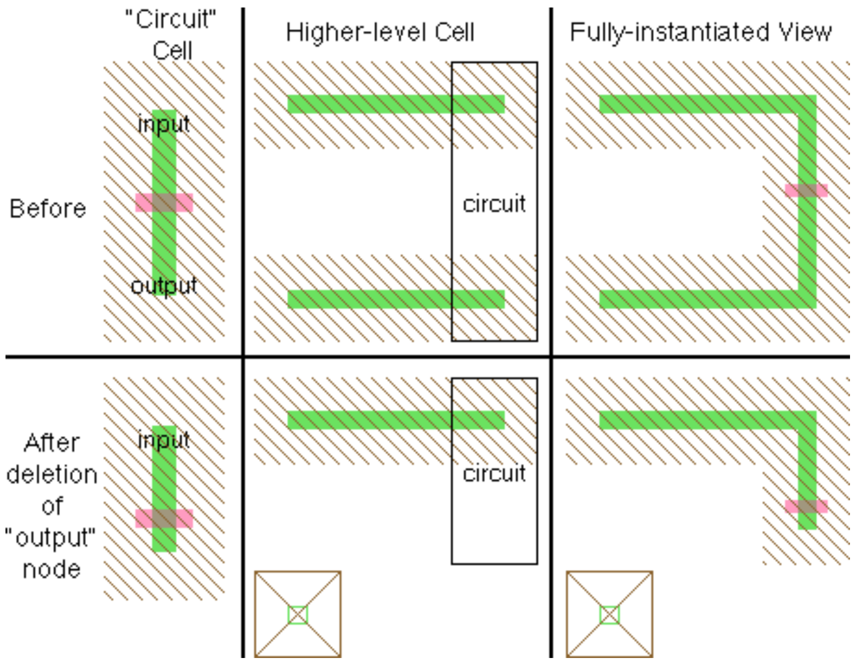
Beginning users often leave many extra pins in their circuits. With the **Cleanup Pins** command (in menu **Edit / Cleanup Cell**), these pins are automatically removed from your circuit, leaving a cleaner network. The command does other pin organizations, such as making sure that text on these pins is located correctly, identifying zero−sized pins, and identifying oversized pins. The **Cleanup Pins Everywhere** command does this function for all cells at once.

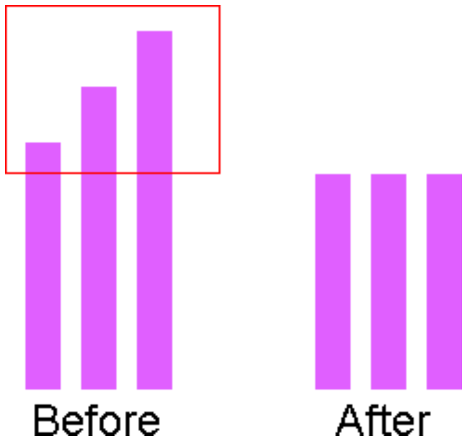
2−3: Circuit Deletion

To remove circuitry, select nodes and/or arcs and use the **Erase** command (in menu **Edit**). A keyboard shortcut for this is the Delete key. If there is a highlighted area rather than a highlighted object, everything in the area is erased.

Note that an arc always connects two nodes, and therefore it cannot remain if one of the nodes is gone. This means that certain rules apply to circuit deletion:

- When a node is erased, all connecting arcs are also deleted. However, if a node is deleted that has exactly two arcs, connected as though the node were in the middle of a single arc, then the node and two arcs are replaced with a single arc.
- In the interest of cleanliness, if an arc is erased, any isolated pins are also erased.
- If an erased node has an export on it (as in this example), then the export disappears and so do all arcs connected to the port on instances of the current cell (for more information on hierarchy, see Chapter 3).





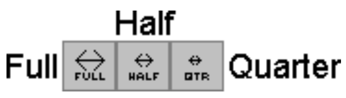
When an area is selected (instead of objects) the **Erase** command erases all geometry in the highlighted area. All arcs that cross into that area will be truncated. Thus, this command truly erases geometry, independent of the structure of nodes and arcs. Note that the area to be erased is adjusted by the current alignment values (see Section 4−7−2). For more on area selection, see Section 2−1−3.

2−4−1: Movement

Components can be moved by clicking on them with the *left* button and then dragging them around while keeping the button pressed. During the drag, the new location of the components will be shown (as well as the amount of motion), and once the button is released the circuitry will be moved.

Another way to move objects is to use the arrow keys. When a node or arc is selected, each press of an arrow key moves that object by one grid unit. If the shift key or the control key is held, then the arrow keys move the object by a block of grid units. A block of grid units is defined in the "Grid" preferences (in menu **File / Preferences...**, "Display" section, "Grid" tab) to be the frequency of bold dots in the grid, initially 10. If you hold both the shift key and the control key, then the distance moved will be a block squared (i.e. initially 100).

The distance that the arrow keys move is also affected by the **Movement** commands (in menu **Edit / Modes**). These commands are also available in the tool bar.



The **Quarter** command causes the amount to be quartered (so unshifted arrow keys will move by a quarter unit). The **Half** command causes the amount to be halved (so unshifted arrow keys will move by a half unit). The **Full** command causes the amount to be full (so unshifted arrow keys will move by one unit). Note that these menu items are attached to the "q", "h", and "f" keys and are also in the tool bar.

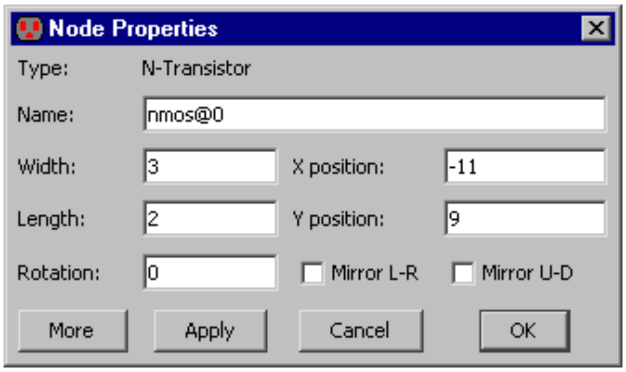
To move objects along only one line (just horizontally or vertically but not both), hold the Control key down during motion. Note that holding the Control key down before clicking will change the nature of the mouse action, so you must click first, and then press Control. When editing schematics, this will constrain objects to movement along 45 degree angles.

When arcs are moved by a large amount, they cause the connecting nodes to move with them. However, for small arc motion, the arc may shift within its ports. This can only happen if the port has nonzero area and if the arc has the *slidable* constraint (shown with the letter "S" when highlighted). These constraints are discussed in greater detail in Section 5−2−2.

2−4−2: Other Modification

Another way to move a node is to use the **Object Properties...** command (in menu **Edit / Properties**), and type new X and Y positions. This dialog allows other modifications to be made as well (orientation, etc.)

The dialog shows the location of the anchor−point of the node.


Node Properties

Type: N-Transistor

Name: nmos@0

Width: 3 X position: -11

Length: 2 Y position: 9

Rotation: 0 ☐ Mirror L-R ☐ Mirror U-D

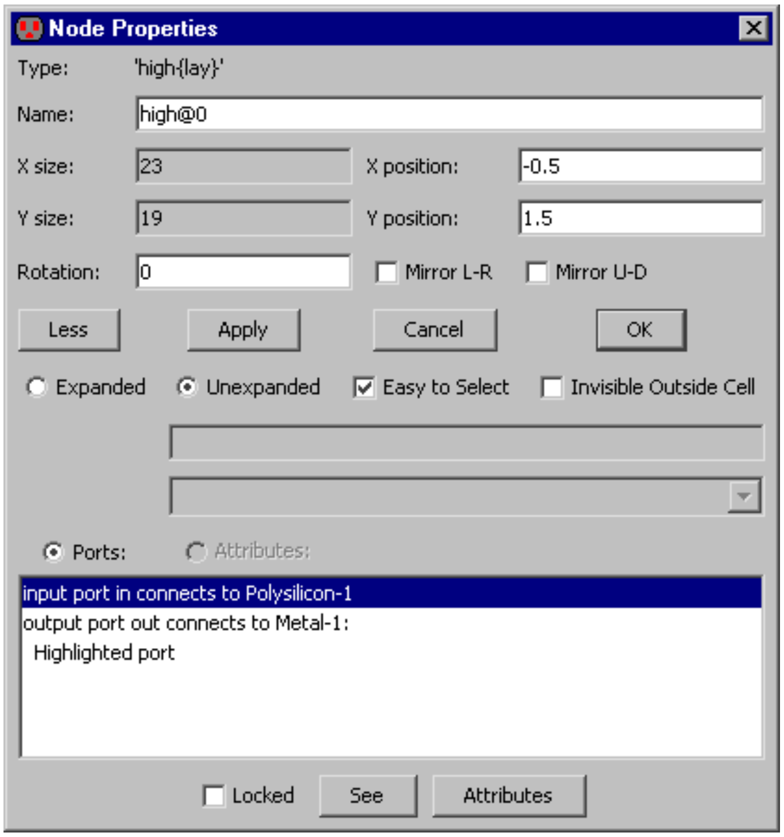
The dialog also has a field for the node's name. This name is not related to network information, but it can be used for identification. If a schematic node is given an arrayed name (such as "and[0:3]") then it indicates that the node is arrayed that many times. Nodes (and arcs) are automatically be given unique names (such as "nmos@0").

This dialog is modeless: it can remain on the screen while other editing is being done. If a different node is selected, the dialog updates to show that node's information. The "Apply" button changes the selected node to match the new values typed into the dialog.

The **Object Properties...** dialog can also expand to show more information. When the "More" button is clicked, it grows to full size as shown.

The full size **Object Properties...** dialog has many new controls, which vary according to the type of node selected:

- "Expanded" and "Unexpanded" control how the node is drawn (if it is a cell instance). An expanded instance is one that shows its contents; an unexpanded instance is drawn as a black box (see Section 3−4).
- "Easy to Select" sets whether this node is selectable with a simple click. This feature allows you to eliminate pieces of circuitry from active editing (see Section 2−1−5).


Node Properties

Type: 'high{lay}'

Name: high@0

X size: 23 X position: -0.5

Y size: 19 Y position: 1.5

Rotation: 0 ☐ Mirror L-R ☐ Mirror U-D

Expanded [✓ Unexpanded] ✓ Easy to Select ☐ Invisible Outside Cell

[✓ Ports:] Attributes:

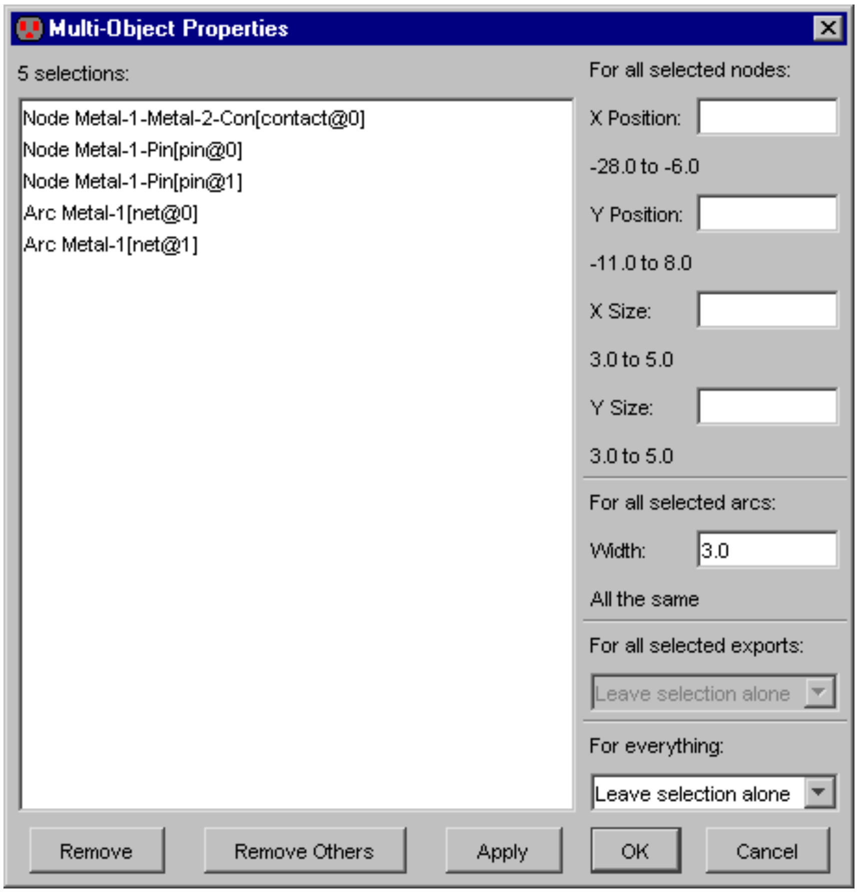
input port in connects to Polysilicon-1
output port out connects to Metal-1:
Highlighted port

☐ Locked

- "Invisible Outside Cell" indicates that this node will not be drawn when the current cell is viewed from higher−up the hierarchy.
- "Locked" nodes may not be changed (moved, deleted).

The bottom of the expanded **Object Properties...** dialog has a scroll area that can view "Ports" or "Attributes". By default, a list of the node's ports is shown, including any exports, connections, and highlight details. If the "Attributes" button is selected, the list shows the attributes on the node. When "Attributes" is selected, the entries in the list let you modify individual values. Note that there is also an "Attributes" button, which brings up a full dialog for editing them.

If many objects are selected, you can move them by a specific distance with the **Move Objects By...** command (in menu **Edit / Move**).



If many nodes are selected, the **Properties...** command will list all of them, and allow position and size changes to be made at once to each in the group. If a position and size value appears in the dialog, it means that this value is the same on every selected node. If the field is blank, it means that there are different values.

Changes are only made in the fields where you type a value. To remove an item from the list (not the circuit, just this list) use the "Remove" button. To remove all but the selected item, use "Remove Others". If only two objects are selected, this dialog shows the distance between their centers.

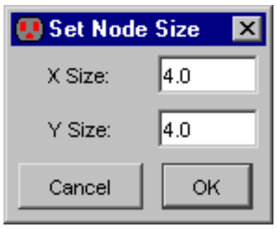
The multi−object **Properties...** dialog also allows you to change selection ease with the "For everything:" popup (see Section 2−1−5 for more on selection styles). When many exports are selected, the dialog allows you to change their characteristics with the "For all selected exports:" popup (see Section 3−6−1 for more on exports).

2−5−1: Node Sizing

To change the size of a node, select it and use the **Interactively** command (in menu **Edit / Size**). After you do that, your mouse movements will stretch an outline around the node. The outline is anchored at the corner farthest from the cursor, and stretches the corner closest to the cursor. An alternate way to resize is to anchor the center of the node and stretch the closest corner to the cursor (with the other three corners adjusting similarly). To get this alternate resizing, press the Shift key while the mouse is pressed.

If you hold the mouse button down while dragging, you can see the final size of the node. Release the mouse button to actually resize the node. To abort this operation, type Escape.

To constrain sizing so that only one dimension changes, hold the Control key while moving the mouse.



X Size: 4.0

Y Size: 4.0

To change the size of more than one node at a time, select the nodes and use the **All Selected Nodes...** command (in menu **Edit / Size**). The dialog allows you to set the X and Y sizes of the selected nodes. If you leave one of these size fields empty, that coordinate is not changed.

Note that when typing size amounts into a dialog, specify the size of the highlighted area. In a typical MOS transistor, the highlighted area (where active and polysilicon cross) is 2x3, even though the component is much larger if you include the four overlap regions sticking out.

2−5−2: Arc Sizing

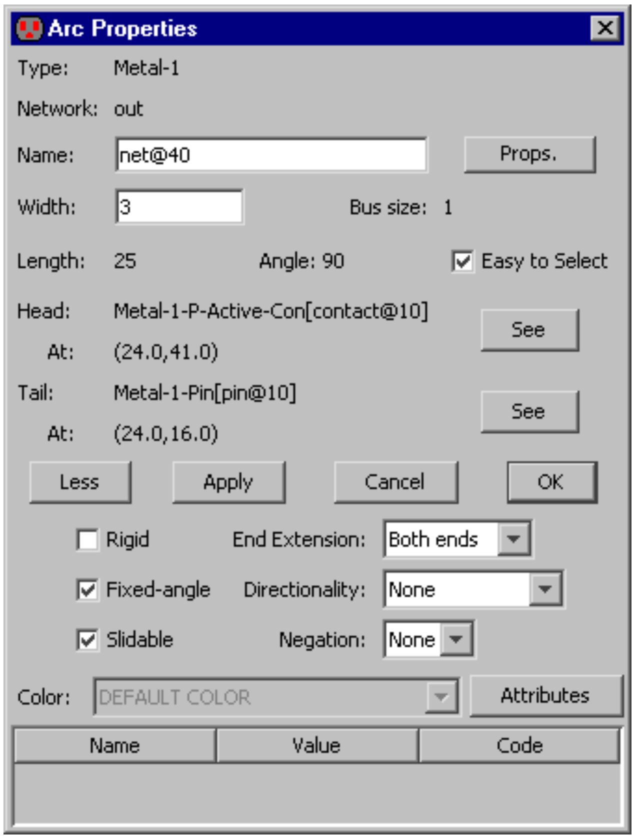
To change the width of an arc, issue the **Interactively** command (in menu **Edit / Size**). Note that the arc stretches about its center so that an edge is at the cursor location. Click a button to make the change. To change the size of more than one arc at a time, select the arcs and use the **All Selected Arcs...** command.

Another way to change an arc's width is to select it and use the **Object Properties...** command (in menu **Edit / Properties**).

Note that when typing size amounts into a dialog, specify the size of the highlighted area. A CMOS active arc shows highlighting only on its active area, even though the complete arc has implant regions that are much larger.

The "Name" field lets you name an arc (see Section 6−8−1). Arc names are only displayed on the arc if they have been explicitly typed into this dialog. You can also use the "Props." button to show a dialog that controls all aspects of a displayed arc name.

The "Easy to Select" checkbox enables selection of the arc with a simple click (see Section 2−1−5).


Arc Properties

Type: Metal-1

Network: out

Name: net@40

Width: 3 Bus size: 1

Length: 25 Angle: 90 [✓] Easy to Select

Head: Metal-1-P-Active-Con[contact@10]

At: (24.0,41.0)

Tail: Metal-1-Pin[pin@10]

At: (24.0,16.0)

[ ] Rigid End Extension: Both ends

[✓] Fixed-angle Directionality: None

[✓] Slidable Negation: None

Color: DEFAULT COLOR

Name	Value	Code

Many pieces of state can be changed here, including Rigid and Fixed−angle (see Section 5−2−1), Slidable (see Section 5−2−2), Directionality (see Section 5−4−1), Ends extension (see Section 5−4−3), and Negation (see Section 5−4−2).

When an Artwork arc has been selected (see Section 7−6−1), the "Color" field is available for setting its color.

2−6: Changing Orientation

There are two commands that can be used to change the orientation of a node. The **Rotate** command (in menu **Edit**) has a submenu that allows the currently highlighted objects to rotate in any of three Manhattan directions or by an arbitrary amount.

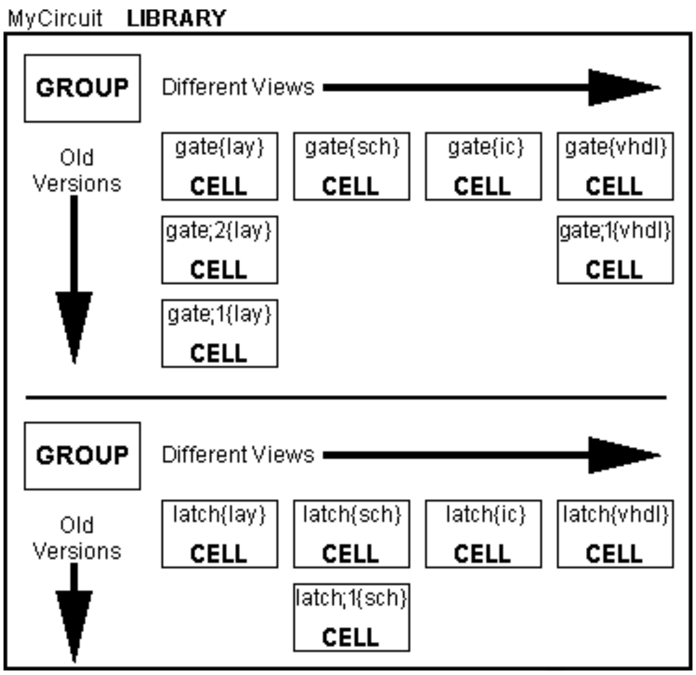
The **Mirror** command (in menu **Edit**) has a submenu that allows you to flip the currently highlighted objects about their vertical centerline (left/right mirroring) or their horizontal centerline (up/down mirroring).

Be aware that mirroring is not the same as rotating, even though both may produce the same visual results. Mirroring causes the node to be flipped about its horizontal or vertical centerline, and thus appear backwards.

Chapter 3: Hierarchy

3−1: Cells

A collection of nodes and arcs is called a *cell*, and instances of cells can be placed in other cells. When a cell instance is placed, that instance is also a node, and is treated just like the simpler transistor and contact nodes. Thus, nodes come in two forms: *primitive* and *complex*. Primitive nodes are found in the component menu and are pre−defined by the technologies (transistors, contacts, pins). Complex nodes are actually instances of other cells, and are found in libraries.

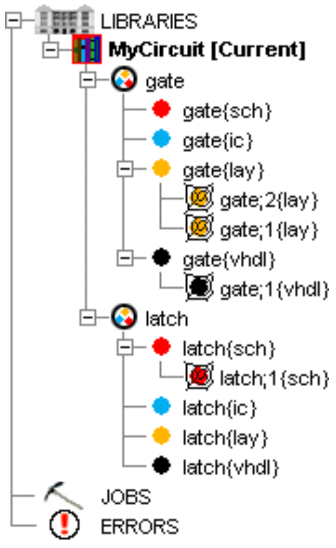


Besides organizing cells into a hierarchy, Electric also organizes cells into *cell groups* and gives each cell a *view* and a *version*. A cell's view describes its contents (for example "layout", "schematics", "netlist", etc.) A cell's version defines its design age. The full name of a cell is:

CELLNAME;VERSION{VIEW}

where CELLNAME is the name of the cell, VIEW is the abbreviated name of this cell's view, and VERSION is the version number of this view of the cell. When no version number is displayed, it implies that this cell is the most recent version (has the largest number). Thus, the cell "gate;2{lay}" is more recent than "gate;1{lay}" but less recent than "gate{lay}" (which must have a higher version number, probably 3).

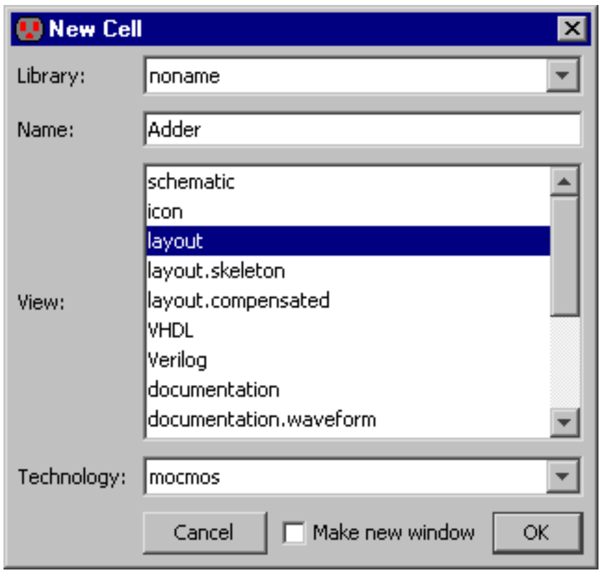
In this example, there is a library with two cell groups. One group has a set of cells called "gate" and the other has a set of cells called "latch". On the right is the explorer view of these cells. See Section 4−8 for more on the cell explorer.



Although it is not necessary for cells in a group to all have the same name, the system presumes that common names will be grouped together. Once in a group, you can rename a cell to give it a different name than the others in its group. Use the **Rename Cell...** command (in menu **Cell**). You can also use context menus in the cell explorer to rearrange groups.

3−2: Cell Creation and Deletion

Cells are created with the **New Cell...** command (in menu **Cell**).



Library: noname
Name: Adder
View: schematic, icon, [✓ layout], layout.skeleton, layout.compensated, VHDL, Verilog, documentation, documentation.waveform
Technology: mocmos

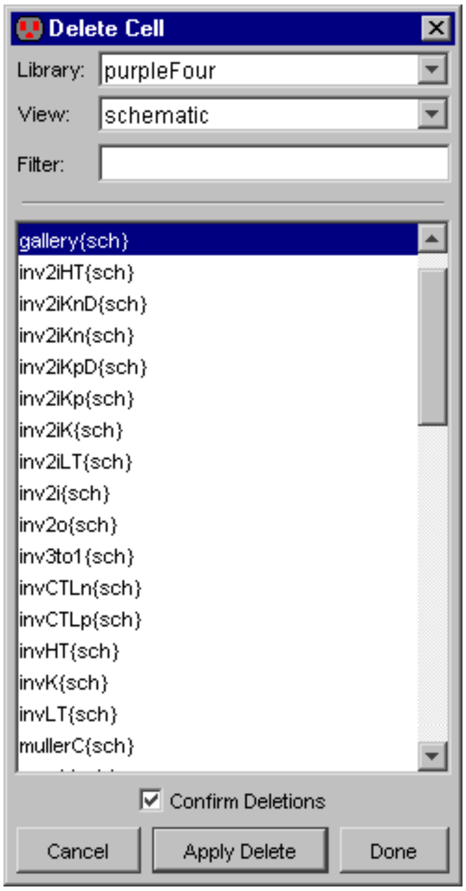
The **New Cell...** command requests a new cell name as well as its view and technology. You can choose to show the cell in the current window, or create a new one.

Cell names may not contain spaces, tabs, unprintable characters, or a colon. Uppercase and lowercase characters are not distinguished: The cell "UPPER" is the same as the cell "Upper." However, the form of capitalization that is used when a cell is first created is retained for all further use.

Another way to create a new cell is to make a copy of an existing one. The **Duplicate Current Cell** and **Duplicate Cell...** commands (in menu **Cell**) copy a cell to a different one with a new name (you will be prompted for the new name). The **New Version of Current Cell** command makes a copy of the cell in the current window, but since it is a "new version", it has the same cell name. The newly created cell is displayed in the window.

Once cells are created you can edit them with the **Edit Cell...** command (in menu **Cell**). Cells can also be edited by using the cell explorer (see Section 4−8 for more).

To delete a cell, use the **Delete Cell...** command (in menu **Cell**). When deleting a cell, there cannot be any instances of this cell, or the deletion fails. As a side effect of failure, you are shown a list of all other cells that have instances of this, so you can see the extent of its use. To find out whether a cell is being used elsewhere in the hierarchy, use the **List Cell Usage** command (in menu **Cell / Cell Info**).

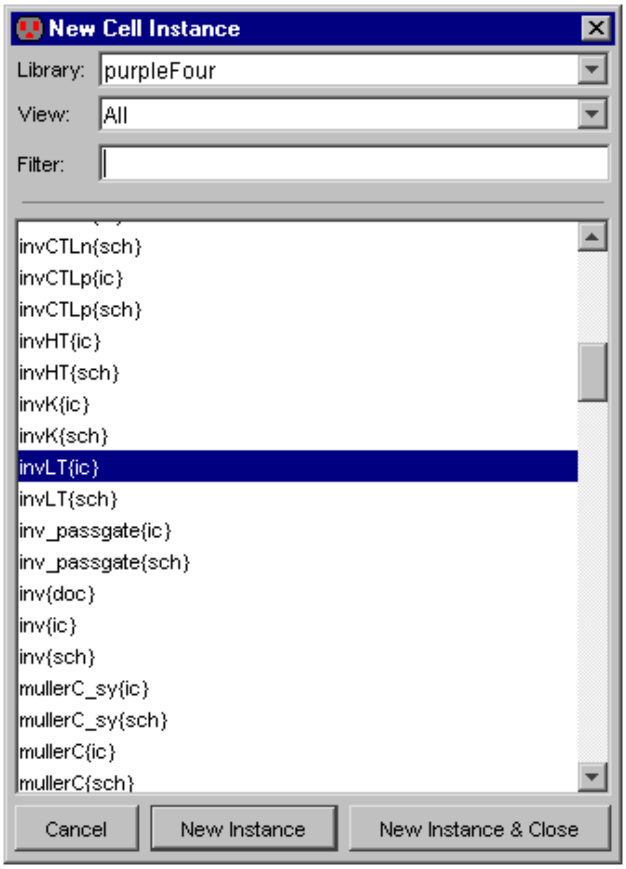


Because Electric is able to keep old versions of cells, deleting the latest version will cause an older version to become the "most recent". Old versions are those whose cell names include the ";VERSION" clause indicating that there is a newer version of this view of the cell. For example, if you have cell "Adder" and an older version "Adder;1", then deleting "Adder" will cause "Adder;1" to be renamed to "Adder". This might make you think that the deletion failed, because there is still a cell called "Adder", but this cell is actually the older (but now most recent) version.

To clean−up old and unused versions of cells, use the **Delete Unused Old Versions** command (in menu **Cell**). Any such cells that are no longer used as instances in other cells will be deleted from the library. You will get a list of deleted cells, and it is possible to undo this command.

3−3: Creating Instances

To place an instance of a cell in another cell, use the "Cell" button in the component menu. After choosing a cell from the popup list, click in the edit window to place the instance.



Library: purpleFour

View: All

Filter:

invCTLn{sch}
invCTLp{ic}
invCTLp{sch}
invHT{ic}
invHT{sch}
invK{ic}
invK{sch}
invLT{ic}
invLT{sch}
inv_passgate{ic}
inv_passgate{sch}
inv{doc}
inv{ic}
inv{sch}
mullerC_sy{ic}
mullerC_sy{sch}
mullerC{ic}
mullerC{sch}

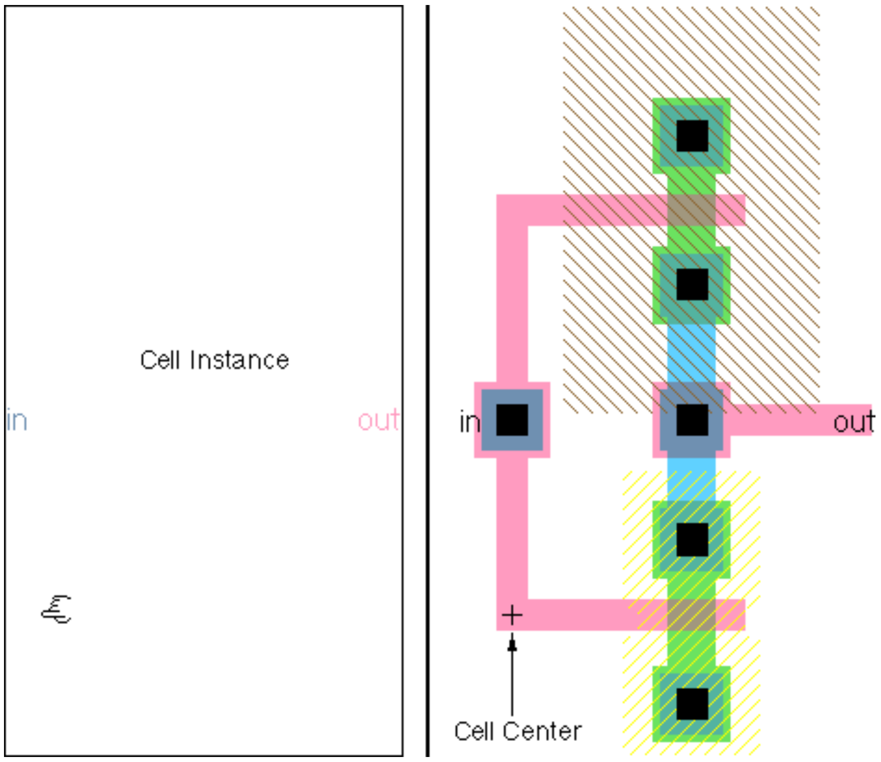
Another way to place an instance of a cell is to use the **Place Cell Instance...** command (in menu **Cell**). You will be shown a list of cells that are available for creation. After selecting one, click to create an instance in the current cell.

The cell selection dialog has three controls at the top for viewing cells. The "Library" popup lets you choose which library to examine. You can choose "ALL" to see cells from all libraries. The "View" popup lets you see only those cells in the specified view. Again, you can choose "All" to see all views. The "Filter" field contains a regular expression that must match a cell name in order to list it.

If you place an instance from a different library, that library will be linked to the current one. Linked libraries are read from disk together, and form a single hierarchy that spans multiple files. See Section 3−9−1 for more on libraries.

An alternate way to create a cell instance is to duplicate an existing one on the screen. This requires that an instance of that particular cell already exist. Select the existing cell and use the **Duplicate** command (in menu **Edit**). Then move the cursor to the intended location of the new instance and click to create the copy. Note that this command copies all attributes of the original node including its orientation.

When a cell instance is being created, the cursor points to its *anchor point*. The anchor point is that point inside of the cell where the coordinate space has its origin. This is often defined by the location of a *cell−center* node inside of the cell.



Most cells have a cell−center node placed automatically in them. If there isn't one and you want it, click on the "Misc" button in the component menu on the left, and choose "Cell Center". A cell−center node, placed inside of the cell definition, affects the anchor point for all subsequent creation of instances of the cell.

3−4: Examining Cell Instances

When instances are initially created, they are drawn as black boxes with nothing inside. This form of instance display is called *unexpanded*, as opposed to *expanded* display which shows the actual layout inside the instance. To expand an instance, select it and use the commands of the **Cell / Expand Cell Instances** menu. The **One Level Down** command opens up the next closed level; the **All the Way** command opens up all levels to the bottom; and the **Specified Amount...** lets you type a number of levels of hierarchy to expand. These commands expand all highlighted cells. If a highlighted cell is already expanded, this command expands any subcells inside of the instance, repeatedly down the hierarchy.

Once expanded, a cell instance will continue to be drawn with its contents shown until the commands of the **Cell / Unexpand Cell Instances** command are used. These commands return cell instances to their black−box form, starting with the deepest subcells that are expanded at the bottom of the hierarchy. The **One Level Up** command closes up the bottommost expanded level; the **All the Way** command closes all levels from the bottom; and the **Specified Amount...** lets you type a number of levels of hierarchy to close.

You can also use the expansion (opened eye) and unexpansion (closed eye) icons from the tool bar to expand and unexpand by one level.

The expansion information can also be set or reset by using the **Object Properties...** command (in menu **Edit / Properties**) and clicking on the "Expanded" or "Unexpanded" buttons.

There are times when you want to see the layout inside of a cell instance, but only temporarily. The **Look Inside Highlighted** command (in menu **Cell**) displays everything in the highlighted area, down through all hierarchical levels. This is a one−shot display that reverts to unexpanded form if the window is shifted, scaled, or redrawn.

There is a slight difference in specification between the **Expand Cell Instances** commands and the **Look Inside Highlighted** command. The **Expand Cell Instances** commands affect cell instances only, and thus any instances that are highlighted or in the highlighted area will be completely expanded. The **Look Inside Highlighted** command affects layout display in an area, so only those parts of instances that are inside of the highlighted area will be shown. Thus, the command **Look Inside Highlighted** is more precise in what it expands and can be used, in conjunction with Area selection, to show only a specific part of the circuit (see Section 2−1−3 for more on area selection).

Chapter 3: Hierarchy

3−5: Moving Up and Down the Hierarchy

Each editing window in Electric displays a single cell. Editing changes can be made only to that cell, and not to any subcells that appear as instances. Thus, you may be able to see the contents of a cell instance, but you cannot edit it.

To edit a cell instance, use the **Down Hierarchy** command (in menu **Cell**). This command will descend into the definition of the currently selected cell instance. The contents will appear at the same size and location as the instance, and you will now be able to edit the contents.

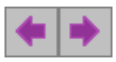
If an icon is selected, the **Down Hierarchy** command will take you to the associated schematic. If the icon that is selected is already in its own schematic (you can place an icon inside its own schematic for documentation purposes), then the **Down Hierarchy** command takes you to the actual icon so that you can edit it.

If a layout cell is selected, you can use the **Down Hierarchy In Place** command to edit the cell while showing the upper level of the hierarchy. The surrounding geometry at the upper level is not editable, and is grayed−out.

Schematic nodes can be arrayed by giving them array names (see Section 6−9−3). When you descend into an arrayed node, the system does not know which element of the array you are entering. Most of the time, the specific element is irrelevant, but if the circuit is being simulated, the specific instance may be necessary for cross−probing. Therefore, if the cell is being simulated and you descend into an arrayed node, you will be prompted for the specific element that you wish to visit. There are other situations that cannot be detected, where the specific element needs to be known. To solve this problem, you can request that Electric prompt for the specific element in all situations where an arrayed node is visited. To do this, check "Always prompt for index when descending into array nodes" in the "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab)

The **Up Hierarchy** command pops you to the next higher cell in the hierarchy. If there was an associated **Down Hierarchy** command, then this returns you to the place where you started, up the hierarchy. If the **Down Hierarchy** commands were not used, Electric attempts to figure out the next higher cell in the hierarchy, switching icons for schematics where appropriate. If there are multiple possibilities (because the current cell is used in many locations) then you will be prompted for a specific location.

Besides using the **Up Hierarchy** and **Down Hierarchy** commands, you can also use the tool bar "back" and "forward" buttons to move through the hierarchy.



If you right−click on these icons, you are given a list of cells and can jump directly to one of them.

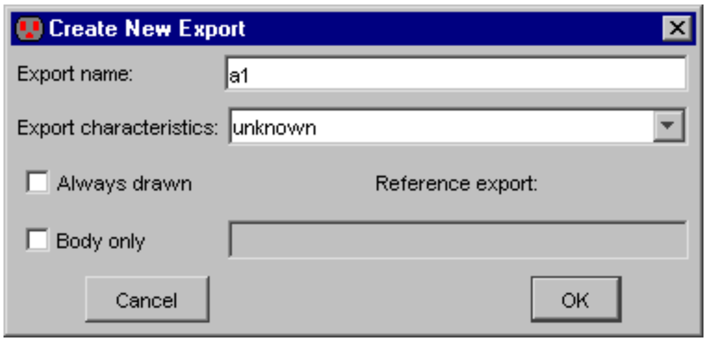
When going down or up the hierarchy, if an export or port is selected, then the equivalent port or export is shown after the level of hierarchy has changed.

3−6−1: Export Creation

All nodes in Electric have connection sites, called *ports*, which indicate where wires may be attached. The primitive nodes have predefined ports, but ports on cell instances must be defined by the user. To do this, simply select a port on a node inside the cell, and turn it into an *export*, which makes it available on all instances of the current cell. Although most ports are on nodes along the edge of the cell, Electric makes no port location restrictions, so they may appear anywhere.

To see the location of all ports on the selected nodes, use the **Show Ports on Node** command (in menu **Export**).

To create an export, select a port on a node and use the **Create Export...** command (in menu **Export**). The resulting dialog requests an export name and some characteristics.



All export names on a cell must be unique; if a nonunique name is given, it is modified to be unique. This modification involves adding "_1", "_2", etc. to the end of scalar export names, or changing the index (from [1] to [2], etc.) for arrayed export names. Like cell names, export names may not contain spaces, tabs, or unprintable characters. Although no case distinction is made between uppercase and lowercase characters; the original case usage is preserved.

Behavioral characteristics can be associated with an export by selecting the appropriate field in the export creation dialog. These behavior characteristics are stored with the export and used primarily by simulators. The characteristics include the following:

- Directional: "input", "output", and "bidirectional".
- Supply: "power" and "ground".
- Clocking: "clock" (a generic clock export) and "clock phase 1" through "clock phase 6".
- Reference: "reference input", "reference output", and "reference base". In addition, reference exports carry an associated export name that is used by the CIF netlister.

The "Always drawn" check box requests that the export label should always appear, regardless of the connection or expansion of its cell. Typically, an export label on an instance of a cell is not displayed when that port is connected to an arc or when the instance is expanded. This check box overrides the suppression.

Another special check box, "Body only," requests that this export not appear when an icon is generated for the cell. This is useful for power and ground exports or duplicate connection sites on a single network.

There are many special exporting commands that are primarily used in array−based layout. If a cell instance is replicated many times and the instances are wired together, then ports on the edge of the array are the only ones that are not wired. These ports define the connections for the next level of hierarchy. What you want to do is to create exports for all unwired ports on all cell instances. To do this, use the **Re−Export Everything** command (in menu **Export**), which generates unique names as it exports all unwired ports on cell instances. To do this same function, but only on the currently highlighted nodes, use **Re−Export Selected**. To do this same function, but inside of the highlighted area, use **Re−Export Highlighted Area**. These two commands can also be run so that wired ports are re−exported, by using the **Re−Export Selected, With Wired Ports** and **Re−Export Highlighted Area, With Wired Ports** To do this same function, but only for Power and Ground exports, use **Re−Export Power and Ground**. Note that ports on primitive nodes are not exported with these commands. See Section 6−4 for more about arrays, and see Section 9−6−1 for more on automatic wiring.

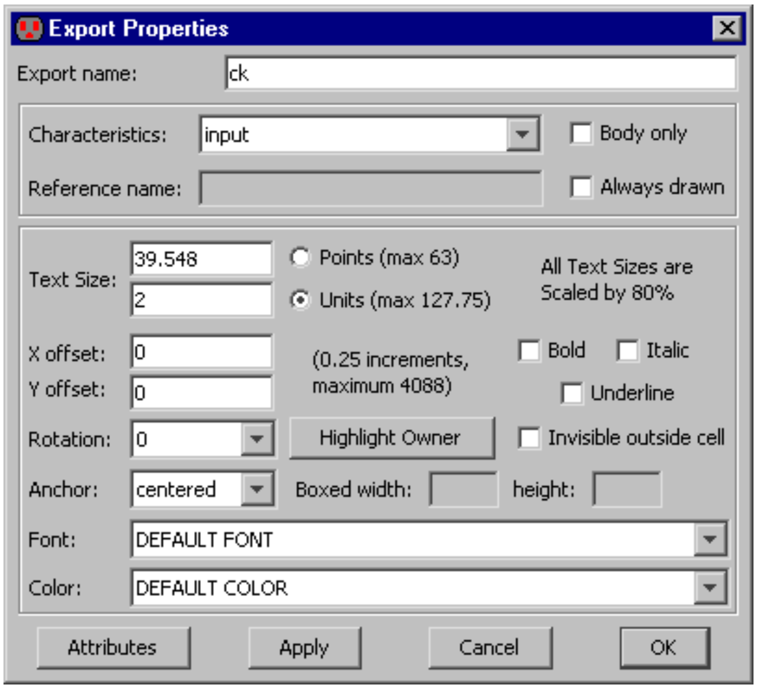
Another special command for export creation is **Add Export from Library...**, which copies exports from another library to the current one. The other library is examined for cells whose names match ones in the current library. When a cell is found in the other library, all of its exports are copied to the cell in the current library (if they don't already exist) and placed in the same location. This command is useful in managing standard cell libraries that are imported from other file formats (see Section 3−9−4 on Standard Cell Libraries). Because some formats contain geometry and others contain connectivity, this command is needed to put them together.

3−6−2: Export Information

Exports are selected by clicking on their text, or by clicking on the node from which they are exported. If a very dense design makes export selection hard, you can choose from a list by using the **Select Object...** command (in menu **Edit / Selection**).

To see all exports that have been defined in the current cell, use the **Show Exports** command (in menu **Export**). The **List Exports** command gives the same information, but in text form, and the **Summarize Exports** command gives a text list that is reduced where sensible. To see a list of exports that are electrically connected to the current object, at multiple levels of hierarchy, use the **List Exports on Network** and **List Exports below Network** commands (in menu **Tool / Network**).

Once a port has been exported, its characteristics can be modified by selecting the export name and using the **Object Properties...** command (in menu **Edit / Properties**). You can change basic export information such as the name, characteristic, and reference name (if applicable). You can control export state such as whether it is always drawn, and whether or not it appears on icons. You can also change the appearance of the export by editing the size, font, color, style, anchor point, and rotation of the name. See Section 6−8−1 for more about text appearance.

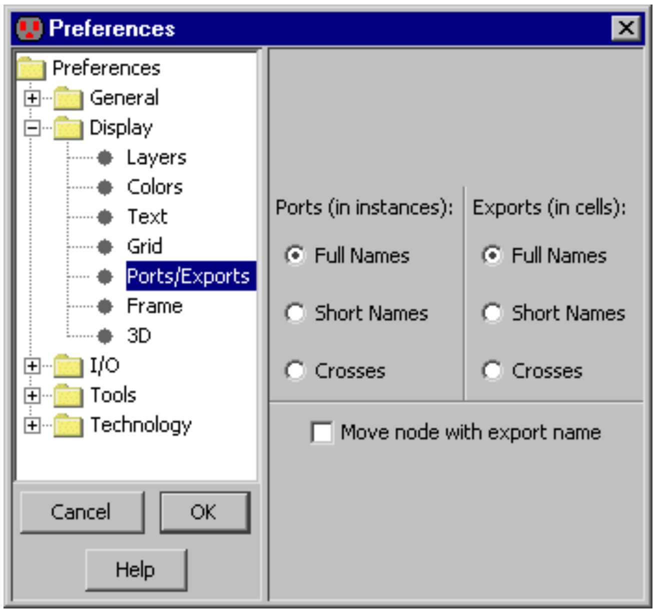


Export name: ck
Characteristics: input
Reference name:
Text Size: 39.548 / 2 [✓ Units (max 127.75)]
X offset: 0
Y offset: 0
Rotation: 0
Anchor: centered
Boxed width:
height:
Font: DEFAULT FONT
Color: DEFAULT COLOR

Special buttons in the Export Properties dialog allow you to examine related objects. The "Highlight Owner" button shows the node on which this export resides, and the "Attributes" button brings up an Attributes dialog for the export (see Section 6−8−5 for more on Attributes).

You can change the characteristics of many exports at once by selecting them and using the **Object Properties...** command (in menu **Edit / Properties**). This multi−object dialog has a "For all selected exports:" popup that will change all export characteristics at once. You can change the name of exports by using the **Rename Export...** command (in menu **Export**).

Ports and exports can be displayed on the screen in many different ways. To control this, use the "Ports/Exports" preferences (in menu **File / Preferences...**, "Display" section, "Ports/Exports" tab). The dialog offers three options for ports and exports: "Full Names" shows full text names, "Short Names" shows port and export names only up to the first nonalphabetic character, and "Crosses" shows crosses at the locations.


Preferences

Preferences
General
Display
Layers
Colors
Text
Grid
Ports/Exports
Frame
3D
I/O
Tools
Technology

Ports (in instances):
[✓ Full Names]
Short Names
Crosses

Exports (in cells):
[✓ Full Names]
Short Names
Crosses

☐ Move node with export name

With short names, the exports "Power−left" and "Power−1" are both written as "Power," which allows multiple exports with the same functionality but different names to be displayed as if they have the same name. To remove port display completely, use the "Layers" tab of the side bar (see Section 4−5). In this panel are options to make exports text completely invisible.

3−6−3: Export Deletion and Movement

You can delete an export simply by selecting its name and using the **Erase** command of the **Edit** menu (or typing the Delete key). You can also use the **Delete Export** command (in menu **Export**).

To remove many exports at once, the **Delete Exports on Selected** command removes all exports on all highlighted nodes. Also, the **Delete Exports in Highlighted Area** command removes only those exports that are in the selected area . When an export is deleted, all arcs connected to that port on instances of the current cell (higher up the hierarchy) are also deleted.

To move export text, simply select it and drag it. The location of the text has no effect on the location of the export: moving the text is only for improvement of the display. However, if you check "Move node with export name" in the "Ports/Exports" preferences (in menu **File / Preferences...**, "Display" section, "Ports/Exports" tab), then moving an export name will cause the node (and the export) to move as well.

It is sometimes desirable to keep an export but to transfer it to another node. If a cell is in use higher in the hierarchy, unexporting and then reexporting deletes all existing connections. Instead, the **Move Export** command (in menu **Export**) can be used. Before using this command, two nodes and their ports must be highlighted with *left* button and *shift−left* button. The export is moved from the first node to the second node.

3−7−1: Cell Lists

To get some basic information about the current cell (size, dates, etc) use the **Describe this Cell** command (in menu **Cell / Cell Info**).

To get information about more than one cell, use the **General Cell Lists...** command. The dialog selects a subset of the cells in the current library.

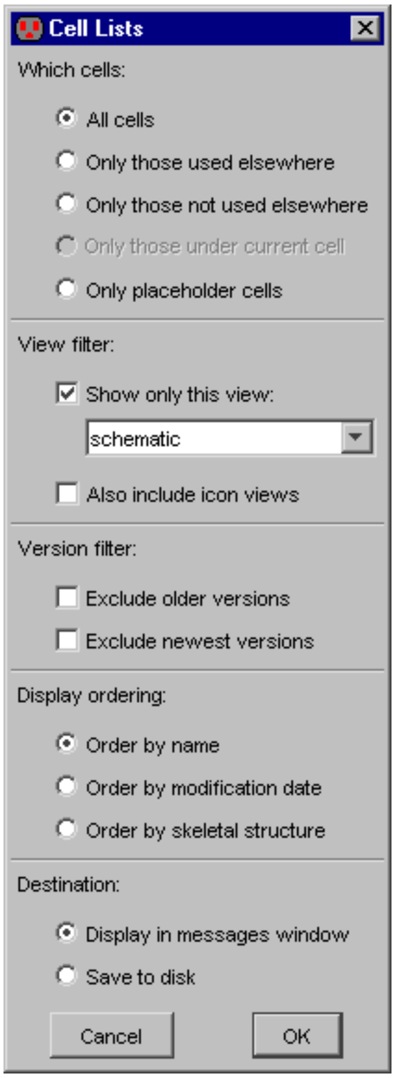
The section labeled "Which cells:" selects the cells to be listed (all, only those used in other cells, only those NOT used in the current cell, only those in the current cell, or only "placeholder" cells: those created because of cross−library dependency failures, see Section 3−9−1).

The section labeled "View filter:" allows only certain views to be displayed.

The section labeled "Version filter:" allows removal of older or newer versions of cells.

The section labeled "Display ordering:" controls the order in which the selected cells will be listed.

The section labeled "Destination:" allows you to dump this listing to a disk file, formatted for spreadsheets (tab−separated).


Cell Lists

Which cells:
[✓ All cells]
Only those used elsewhere
Only those not used elsewhere
Only those under current cell
Only placeholder cells

View filter:
[✓ Show only this view:] schematic
Also include icon views

Version filter:
Exclude older versions
Exclude newest versions

Display ordering:
[✓ Order by name]
Order by modification date
Order by skeletal structure

Destination:
[✓ Display in messages window]
Save to disk

The result of cell information listing looks like this:

```
-Cell------------Version----Creation date----------Revision Date---------Size----Usage--L-I-C-D
tech-Artwork{}        1  Dec 31, 1969 16:00:00  Dec 15, 2004 11:34:15  131.0x83.0   0    L
tech-Bipolar{ic}      1  Dec 15, 2004 11:34:25  Dec 15, 2004 11:34:25   10.0x12.0   1
tech-Bipolar{lay}     1  Jul 23, 1990 23:25:49  Dec 15, 2004 12:38:11   37.0x73.5   0
tech-Bipolar{sch}     1  Jul 26, 1990 23:58:58  Dec 15, 2004 11:34:27  58.75x59.5   0    L I
tech-DigitalFilter{} 1  Dec 31, 1969 16:00:00  Dec 01, 2000 13:56:47   48.0x45.5   0
tech-MOSISCMOS{lay}  1  Jul 24, 1998 16:10:55  Dec 09, 2001 12:35:29   85.5x83.0   0          D
tech-PCB7404{}        1  Dec 31, 1969 16:00:00  Dec 15, 2004 11:45:03   12.5x28.5   1
tool_NCC{sch}         1  Mar 27, 2001 06:35:49  Jan 25, 2002 15:57:57   44.0x41.5   0    L I
```

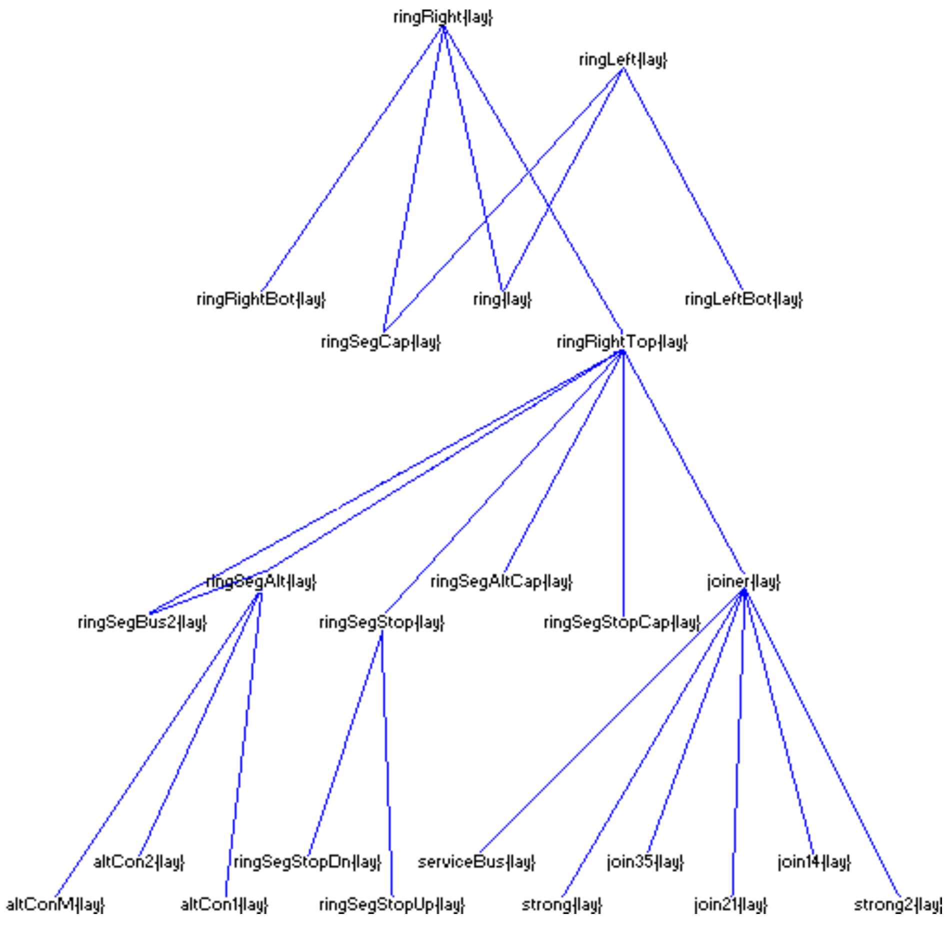
The last five columns show the usage and four state bits. The usage is the number of times that this cell appears as an instance in other cells. The state bits are:

- "L" if the cell contents are locked
- "I" if instances in the cell are locked
- "C" if the cell is part of a cell library
- "D" if the cell has passed design−rule checking

For more cell information, use the commands of menu **Cell / Cell Info**. The **List Nodes in this Cell** command shows all nonprimitive nodes in the current cell. The **List Cell Instances** command shows all cell instances below the current cell. The **List Cell Usage** command looks up the hierarchy and finds cells that contain the current cell as an instance.

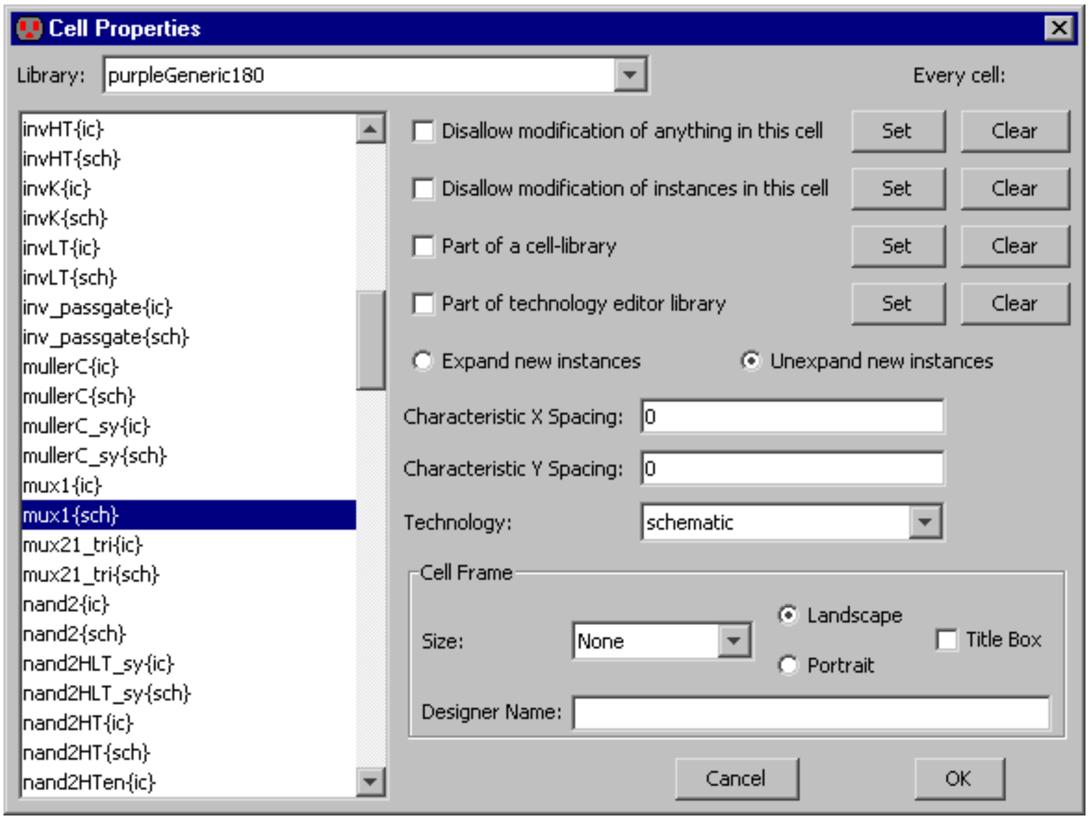
3−7−2: Cell Graphing

The **Graphically, Entire Library** command (in menu **Edit / Cell Info**) displays a graph of every cell in the library. The **Graphically, From Current Cell** command displays a graph that places the current cell at the top. These commands create a graph of the cell hierarchy. This graph is actually a new cell, called "CellStructure", built from Artwork nodes, and stored in the current library.



3−7−3: Cell Properties

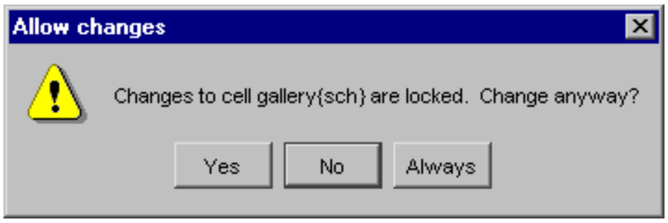
To examine and set more information about existing cells, use the **Cell Properties...** command (in menu **Cell**): The left side of the dialog lists cells by library. On the right are the properties of these cells.



The checkbox "Disallow modification of anything in this cell", allows you to control whether the contents of a cell is editable or not. When modification is disallowed, no changes may be made. This is useful when you want to allow examination without accidental modification.

The checkbox "Disallow modification of instances in this cell", also prevents changes to the selected cell, but in this case, only instances of sub−cells are locked. This is useful when you have a correct instance placement and are doing wiring.

If you make a change that has been disallowed, a dialog appears that asks if you want to override the lock. You may make the change ("Yes"), disallow the change ("No"), or remove the lock ("Always", which unchecks the locks in this dialog).



Changes to cell gallery{sch} are locked. Change anyway?

The check box "Part of a cell−library" indicates that this cell is from a library of standard cells and should be treated accordingly.

The check box "Part of technology editor library" indicates that this cell helps to define a technology. For more on the technology editor, see Section 8−1.

For the first 4 checkboxes in this dialog, there are buttons on the right which allow you to set or clear these flags for all cells in the library.

The "Expand new instances" and "Unexpand new instances" buttons choose whether newly created instances of this cell are expanded (contents visible) or unexpanded (drawn with a black outline) See Section 3−4 for more on expansion.

The "Characteristic Spacing" is the spacing of this cell when arrayed (see Section 6−4).

Each cell is tied to a specific technology. The cell's technology is set when the cell is created. You can change the technology that is associated with a cell by using the "Technology" popup.

The lower−right has frame controls for the cell. The frame is a border that is usually drawn around schematics. You can set the frame size, whether it is wider (Landscape mode) or taller (Portrait mode), and whether a title box is drawn in the corner. Additionally, you can set the designer name to be drawn for each cell. Other information in the title box (company name, project name) are set on a per−user or per−library basis with the "Frame" preferences (in menu **File / Preferences...**, "Display" section, "Frame" tab). See Section 7−5−2 for more on frames.

3−8: Rearranging Cell Hierarchy

In order to manipulate hierarchical circuits, it is useful to create and delete levels of the hierarchy. The **Package Into Cell...** command (in menu **Cell**) collects everything in the highlighted area into a new cell. You will be prompted for the cell name. The most convenient way to specify an area for packaging is to use the Area Selection commands (see Section 2−1−3). Every node touching the area is included in the new cell. All arcs between nodes in the area are also included. The highlighted circuitry is not affected.

The opposite function is the removal of levels of hierarchy. This is done with the **Extract Cell Instance** command, which takes the currently highlighted cell instances and replaces them with their contents. Repeated use of this command goes further down the hierarchy until it is fully instantiated. All arcs that were connected to the cell instances are reconnected to the correct parts of the instantiated circuitry.

Chapter 3: Hierarchy

3−9−1: Introduction to Libraries

A *library* is a collection of cells that forms a consistent hierarchy. To enforce this consistency, Electric stores an entire library in one disk file that is read or written at one time. It is possible, however, to have multiple libraries in Electric. Only one library is the current one, and this sometimes affects commands that work at the library level. When there are multiple libraries, you can switch between them with the **Change Current Library...** command (in menu **File**) or by using the library's context menu in the cell explorer (see Section 4−8). To see which libraries are read in, use the **List Libraries** command.

To create a new, empty library, use the **New Library...** command (in menu **File**). To change the name of the current library, use the **Rename Library...** command. To delete a library, use the **Close Library** command. This removes only the memory representation, not the disk file. Note that library changes are too vast to be tracked by the database−change mechanism and so are not undoable.

It is possible to link two libraries by placing an instance of a cell from one library into another (this is done with the **Place Cell Instance...** command in menu **Cell**). When this happens, the library with the instance (the main library) is linked to the library with the actual cell (this is the *reference library*). Because the reference library is needed to complete the main library, it will be read whenever the main library is read.

If referenced libraries are edited independently, it is possible that a reference to a cell in another library will not match the actual cell in that library. When this happens, Electric creates a "placeholder" cell that matches the original specification. Thus, the link to the referenced library is broken because the cell there does not fit where the instance should be. To see a list of all placeholder cells that were created because of such problems, use the **General Cell Lists...** command (in menu **Cell / Cell Info**) and select "Only placeholder cells".

When reading and writing libraries, Electric offers a number of ways to choose the directory in the file system. The choices are available in the "Working directory" field of the "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab):

- "Based on OS" indicates that Electric should choose a directory based on the standard look−and−feel used in your operating system.
- "Use current directory" indicates that Electric should use the current directory (the one that was current when Electric started).
- "Use last used directory" indicates that Electric should remember the last directory that it used (even across sessions).

Electric comes with some built−in libraries. There are two Spice primitive libraries (see Section 9−4−3). A library of examples can be loaded with the **Load Library** command (in menu **Help / Samples**). A set of MOSIS CMOS pads can be loaded with the **MOSIS CMOS Pads** command (in menu **Help / Load Built−in Libraries**). A set of gates, useful for Logical Effort (see Section 9−9), can be loaded with the **Load Logical Effort Libraries (Purple, Red, and Orange)** command (in menu **Tool / Logical Effort**).

3−9−2: Reading Libraries

The **Open Library...** command (in menu **File**) brings a new library into Electric from disk. These libraries may have the extension ".elib" or ".jelib" (the jelib format is the default, see Section 10−1).

You can also use the open−library icon from the tool bar.

Electric users with very old ".elib" files may have difficulty reading them into Electric. If you have been using versions of Electric prior to 7.00, it may help to upgrade to that version and read the libraries. Saving ".elib" files from version 7.00 will work properly in the current system.

By default Electric searches for libraries in the working directory, absolute file path references, and Electric's internal library directory. Users can specify additional directories to search by using a file called "LIBDIRS" placed in the working directory. This file specifies additional paths to search for library files. The file has the following syntax:

```
* <comments>
include <another_LIBDIRS_file>
<library_directory>
```

Paths may be absolute or relative.

Besides Electric libraries, it is possible to read circuit descriptions that are in other formats with these commands in the **File / Import** menu:

- **CIF (Caltech Intermediate Format)** is used to describe integrated circuit layout. It contains no connectivity, so after the library is read, it does not know about transistors and contacts: just layers. You can use the node extractor to convert CIF to real Electric components (see Section 9−10−2). To affect how CIF is read, use the "CIF" preferences (in menu File / Preferences..., "I/O" section, "CIF" tab). See Section 7−3−2 for more on CIF.
- **GDS II (Stream)** is also used to describe integrated circuit layout. It contains no connectivity, so after the library is read, it does not know about transistors and contacts: just layers . You can use the node extractor to convert GDS to real Electric components (see Section 9−10−2). To affect how GDS is read, use the "GDS" preferences (in menu File / Preferences..., "I/O" section, "GDS" tab). See Section 7−3−3 for more on GDS.
- **EDIF (Electronic Design Interchange Format)** is used to describe both schematics and layout. Electric reads EDIF version 2 0 0. Use the "EDIF" preferences (in menu File / Preferences..., "I/O" section, "EDIF" tab) to affect how EDIF is read (see Section 7−3−4).
- **LEF (Library Exchange Format)** is an interchange format that describes the cells in a library. The cells that are read in contain ports, but very little contents.
- **DEF (Design Exchange Format)** is an interchange format that describes the contents of a library. DEF input often makes use of associated LEF files which must already have been read. Use the "DEF" preferences (in menu File / Preferences..., "I/O" section, "DEF" tab) to affect how DEF is read (see Section 7−3−5).
- **DXF (AutoCAD)** is a solid−modeling interchange format, and so it may contain 3D objects that cannot be read into Electric. Nevertheless, Electric creates a library of artwork primitives as well as it can. Use the "DXF" preferences (in menu File / Preferences..., "I/O" section, "DXF" tab) to affect how DXF is read (see Section 7−3−7).
- **SUE (Schematic User Environment)** is a schematic editor that captures a single cell in each file. The circuitry in SUE files is added to the current library instead of being placed in its own library

(because many SUE files may have to be read to build up a single Electric library). Use the "SUE" preferences (in menu File / Preferences..., "I/O" section, "SUE" tab) to affect how SUE is read (see Section 7−3−8).

- **ELIB** is an older Electric library format that is in an undocumented binary format.
- **Readable Dump** is an older Electric library format that captures the entire database in a text−readable format. These files were used when the ".elib" file was the main way of saving libraries, because a way was needed of reading library files. Now that the newer ".jelib" format is also text−readable, there is no need to use Readable Dumps anymore.
- **Text Cell Contents** is used to read a text file into a text cell. The current window must be a textual view (such as VHDL, Verilog, documentation, etc.) See Section 4−10 for more on text windows.

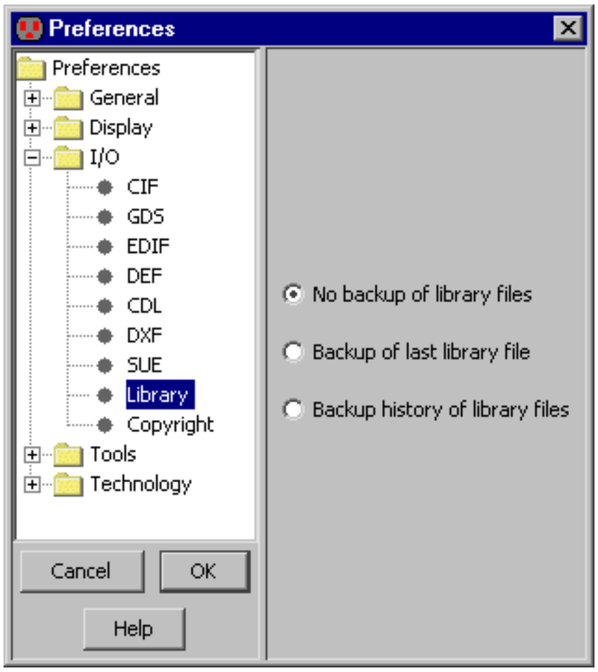
3−9−3: Writing Libraries

Writing libraries to disk is done with the **Save Library** command (in menu **File**).

The **Save All Libraries** command writes all libraries that have changed. You can also use the save−libraries icon from the tool bar. To force all libraries to be saved, use the **Mark All Libraries for Saving** command, or use **Save All Libraries in Format** to specify how they are to be saved.

If a library was read from disk, it is written back to the same file. If, however, you wish to write the library to a new file (thus preserving the original) then use the **Save Library As...** command.

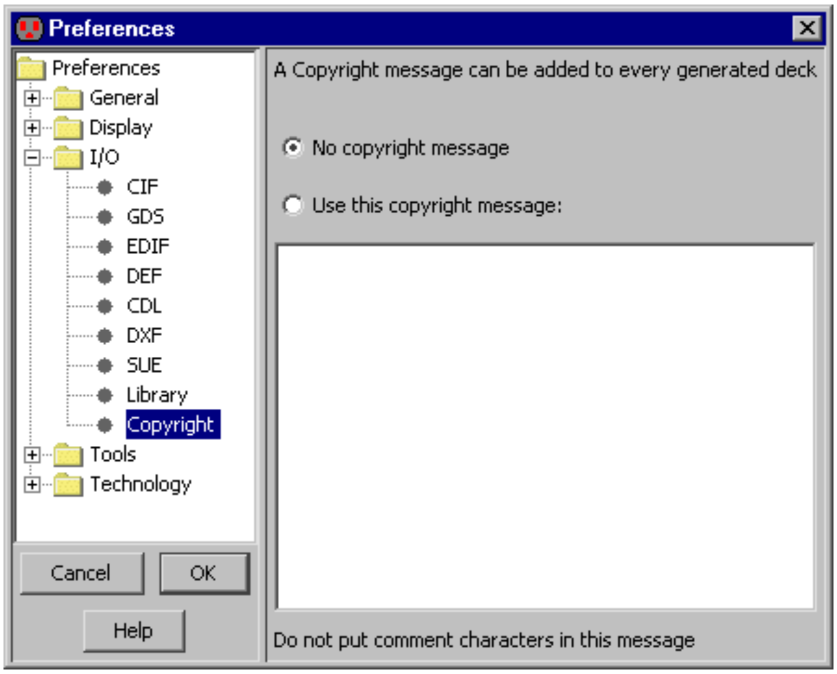
The "Library" preferences (in menu **File / Preferences...**, "I/O" section, "Library" tab) offers options for writing libraries to disk. By default, saved libraries overwrite the previous files and no backup is created. If you choose "Backup of last library file", then the former library is renamed so that it has a "~" at the end. If you choose "Backup history of library files", then the former library is renamed so that it has its creation date as part of its name.



[✓ No backup of library files]
Backup of last library file
Backup history of library files

Electric can also write external format files with these commands in the **File / Export** menu:

- **CIF (Caltech Intermediate Format)** is used to describe integrated circuit layout. The output file contains only the current cell and any circuitry below that in the hierarchy. Use the "CIF" preferences (in menu **File / Preferences...**, "I/O" section, "CIF" tab) to affect how CIF is written. See Section 7−3−2 for more on CIF.
- **GDS II (Stream)** is also used to describe integrated circuit layout. The output file contains only the current cell and any circuitry below that in the hierarchy. Use the "GDS" preferences (in menu **File / Preferences...**, "I/O" section, "GDS" tab) to affect how GDS is written. See Section 7−3−3 for more on GDS.
- **EDIF (Electronic Design Interchange Format)** can write either the Netlist or the Schematic view of the circuit. Electric writes EDIF version 2 0 0. Use the "EDIF" preferences (in menu **File / Preferences...**, "I/O" section, "EDIF" tab) to affect how EDIF is written. See Section 7−3−4 for more on EDIF.
- **LEF (Library Exchange Format)** is an interchange format that describes the exports on cells in a library.
- **L** is the GDT language, still appearing in some commercial systems. The output file contains only the current cell and any circuitry below that in the hierarchy.
- **Eagle** is an interface to the Eagle schematics design system (its netlist format). Before writing Eagle files, you must give every node the "ref_des" attribute, and every port on these nodes the "pin" attribute. If you also place the "pkg_type" attribute on the node, it overrides the cell name.
- **ECAD** is an interface to the ECAD schematics design system (its netlist format). Before writing ECAD files, you must give every node the "ref_des" attribute, and every port on these nodes the "pin" attribute. If you also place the "pkg_type" attribute on the node, it overrides the cell name.
- **Pads** is an interface to the Pads schematics design system (its netlist format). Before writing Pads files, you must give every node the "ref_des" attribute, and every port on these nodes the "pin" attribute. If you also place the "pkg_type" attribute on the node, it overrides the cell name.
- **Text Cell Contents** is used to write a text file from a text cell. The current window must be a textual view (such as VHDL, Verilog, documentation, etc.) See Section 4−10 for more on text windows.
- **PostScript** is the Adobe printing language. The output file contains only a visual representation of the current cell (or part of that cell). PostScript options can be controlled with the "Printing" preferences (in menu **File / Preferences...**, "General" section, "Printing" tab).
- **HPGL** is the Hewlett−Packard printing language. The output file contains only a visual representation of the current cell (or part of that cell).
- **PNG (Portable Network Graphics)** is an image format that captures the current window.
- **DXF (AutoCAD)** is a solid−modeling interchange format. Use the "DXF" preferences (in menu **File / Preferences...**, "I/O" section, "DXF" tab) to affect how DXF is written. See Section 7−3−7 for more on DXF.
- **ELIB (Version 6)** writes old−format binary files. These files can be read by version 6 of Electric.



A Copyright message can be added to every generated deck

[✓ No copyright message]

Use this copyright message:

Do not put comment characters in this message

The exported files from Electric are often considered to be proprietary information, and must be marked appropriately. Copyright information can be inserted into exported files with the "Copyright" preferences (in menu **File / Preferences...**, "General" section, "Copyright" tab). Since each export file has a different format for comments, the copyright text should not contain any such characters. Instead, the system will insert the proper comment characters for the particular export format.

The copyright information will be inserted into decks exported for CIF, LEF, and PostScript, as well as in simulation netlists for Verilog, Spice, Silos, ESIM/RSIM/RNL/COSMOS, FastHenry, Maxwell, and IRSIM.

3−9−4: Standard Cell Libraries

Electric does not come with any useful libraries for doing design. However, the system is able to make use of Artisan libraries. These libraries are free, provided that you sign an Artisan license. Once you are licensed, you will have standard cell libaries, pad libraries, memory libraries, and more.

Artisan libraries are not distributed in Electric format. Instead, they come in a variety of formats that can be read into Electric. The GDS files contain the necessary geometry, and the LEF files contain the connectivity. By combining them, Electric creates a standard cell library that can be placed−and−routed and can be fabricated. Note that the data is not node−extracted, so not all of Electric's capabilities can be used with this data.

To create an Artisan library, follow these steps:

- Select the Artisan data that you want, and extract the GDS and LEF files for it. The GDS files will have the extension ".gds2", which is not what Electric expects (Electric expects them to end with ".gds"), so you may want to rename them.
- Read the LEF file into Electric with the **LEF (Library Exchange Format)...** command (in menu **File / Import**). Keep in mind that the LEF data may come in multiple versions for different numbers of metal layers.
- Read the GDS data into Electric with the **GDS II (Stream)...** command (in menu **File / Import**). Note that the proper GDS layers must be established first (with the "GDS Preferences", see Section 7−3−3). There will now be two libraries in memory: one with the GDS data and one with the LEF data.
- Merge the port information from the LEF library into the GDS library. It is important that the GDS library be the "current library" (use the **Change Current Library...** command in menu **File** if it is

not). To merge the LEF port information, use the **Add Exports from Library...** command (of menu **Export**). You will be prompted for another library, and should select the one with the LEF data.

- At this point, the GDS library now has standard cells in it, including the export information that was in the LEF library. Before saving it to disk, you should probably use the **Cell Properties...** command (of menu **Cells**, see Section 3−7−3) and set all of the cells to be "Part of a cell−library".

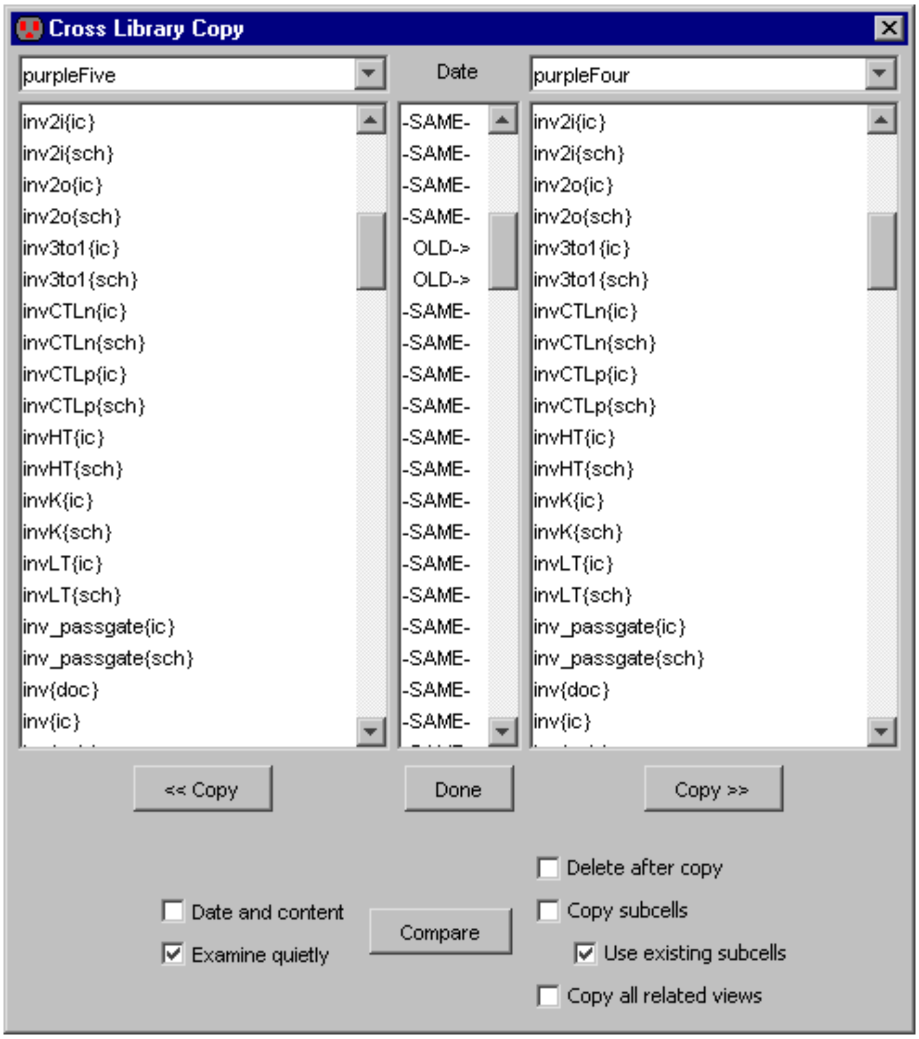
Chapter 3: Hierarchy

3−10: Copying Cells Between Libraries

In general, different libraries are completely separate collections of cells that do not relate. For example, two cells in different libraries can have the same name without being the same size or having the same content. Although a cell from one library can be used as an instance in another, this causes the two libraries to be linked together. It may be simpler to copy the cells from one library to another, thus allowing a single library to contain the entire design.

The **Cross−Library Copy...** command (in menu **Cell**) provides a dialog for copying cells between libraries. The left and right columns show the contents of two different libraries (and the pulldowns above each column let you select the two libraries that you want to use).

When there is a cell with the same name in both libraries, the system compares them to determine which is newer.



purpleFive	Date	purpleFour
inv2i{ic}	-SAME-	inv2i{ic}
inv2i{sch}	-SAME-	inv2i{sch}
inv2o{ic}	-SAME-	inv2o{ic}
inv2o{sch}	-SAME-	inv2o{sch}
inv3to1{ic}	OLD->	inv3to1{ic}
inv3to1{sch}	OLD->	inv3to1{sch}
invCTLn{ic}	-SAME-	invCTLn{ic}
invCTLn{sch}	-SAME-	invCTLn{sch}
invCTLp{ic}	-SAME-	invCTLp{ic}
invCTLp{sch}	-SAME-	invCTLp{sch}
invHT{ic}	-SAME-	invHT{ic}
invHT{sch}	-SAME-	invHT{sch}
invK{ic}	-SAME-	invK{ic}
invK{sch}	-SAME-	invK{sch}
invLT{ic}	-SAME-	invLT{ic}
invLT{sch}	-SAME-	invLT{sch}
inv_passgate{ic}	-SAME-	inv_passgate{ic}
inv_passgate{sch}	-SAME-	inv_passgate{sch}
inv{doc}	-SAME-	inv{doc}
inv{ic}	-SAME-	inv{ic}

[ ] Date and content
[✓] Examine quietly
[ ] Delete after copy
[ ] Copy subcells
[✓] Use existing subcells
[ ] Copy all related views

If you check "Date and content" (and then "Compare" to do comparison again) Electric will compare the actual contents of cells when determining their equality. Unchecking "Examine quietly" will cause the system to describe differences found during comparison.

By choosing a cell in the right−hand library and clicking "<< Copy", that cell is copied into the left−hand library. The "Copy >>" button does the reverse. If "Delete after copy" is checked, the buttons change to "<< Move" and "Move >>".

The system can be requested to copy additional cells that relate to the selected one. By checking "Copy subcells", all subcells of the copied cell are also transfered. By checking "Copy all related views", all related views (icon, schematic, layout, etc.) are also transfered. Note that if "Copy all related views" is off but you want to "Copy subcells", it still copies related views in a limited fashion (i.e. schematics and icons are copied together).

When there is a reference to an instance inside of a copied cell, and that instance already exists in the destination library, there are many ways to handle the transfer. For example, library "Frank" has cell "A" which has, inside of it, an instance of cell "B" ("B" is also in library "Frank"). You want to copy cell "A" to library "Tom", but there is already a cell called "B" in library "Tom". These things may happen:

- If "Copy subcells" is checked, then a new version of "Tom:B" is created from "Frank:B", and this cell is instantiated in the copied "Tom:A".
- If "Copy subcells" is not checked, the instance in the new "Tom:A" points to the old "Frank:B".
- If "Copy subcells" is not checked and "Use existing subcells" is checked, the instance in the new "Tom:A" points to the existing cell "Tom:B". In order for this to work, however, the size and exports of "Tom:B" must match the original in "Frank:B".

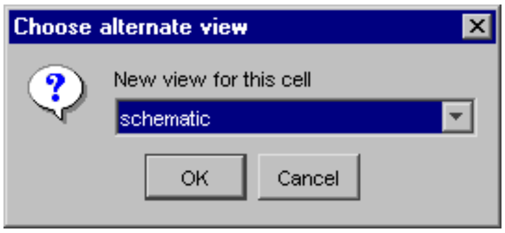
Therefore, if "Copy subcells" is checked, "Use existing subcells" is implied.

3−11−1: Setting a Cell's View

Each cell has a *view*, which provides a description of its contents. A view consists of a full name and an abbreviation to be used in cell naming. For example, the "layout" view is abbreviated "lay" and so the layout view of cell "adder" is called "adder{lay}." When no view name appears, the cell has the "unknown" view. Possible views are:

- "layout" (for IC layout)
- "schematic" (for logic designs)
- "icon" (to describe a cell symbolically)
- "layout.skeleton" (a minimal view)
- "documentation" (a text−only view)
- "VHDL" or "Verilog" (text−only views for hardware−description languages)
- a number of "netlist" views (text−only views that list connectivity for various tools such as "netlisp", "als", "quisc", "rsim", and "silos")
- "unknown" (no specified view)

When creating a cell with the **New Cell...** command, you can specify its view. After creation, you can change the current cell's view with the **Change Cell's View...** command (in menu **View**). You can also use context menus in the cell explorer to change a cell's view. Note that this is one of the few commands in Electric that is NOT undoable.



3−11−2: Switching between Views of a Cell

When editing one view of a cell, there are commands in the **View** menu that will switch to an alternate view of the same cell.

- Use **Edit Layout View** to switch to the layout view.
- Use **Edit Schematic View** to switch to the schematic view.
- Use **Edit Icon View** to switch to the Icon view.
- Use **Edit VHDL View** to switch to the VHDL view.
- Use **Edit Documentation View** to switch to the text−only documentation view.
- Use **Edit Skeleton View** to switch to the Skeleton view.

For all other view types, use **Edit Other View...** and select the desired view. Note that these commands are equivalent to the **Edit Cell...** command (in menu **Cell**) with an appropriate selection.

When editing cells with text−only views (VHDL, Documentation, etc.), the window becomes a text editor. You may then use the **Text Cell Contents...** commands (in menu **File / Export** and **File / Import**) to save and restore this text to disk. See Section 4−10 for more on text editing.

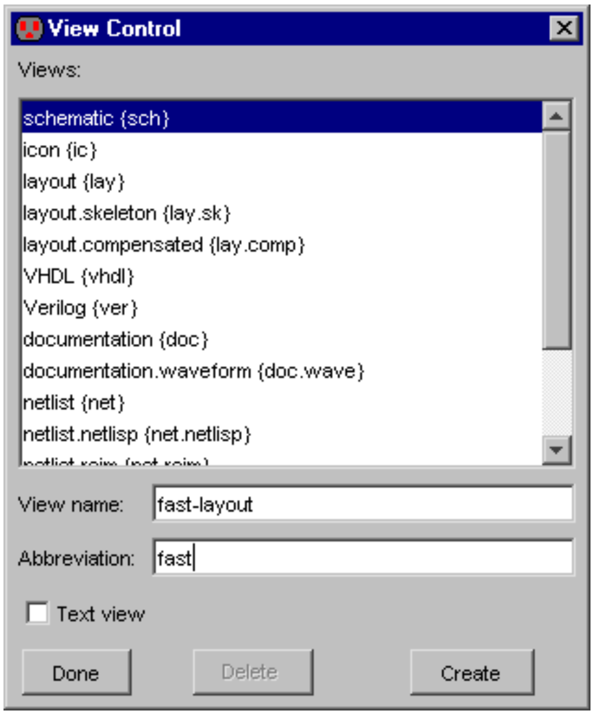
The commands to edit another view work only when that cell exists. To create a new cell of a particular type, use the **Make...** commands of the **View** menu. These view conversion commands are available:

- **Make Icon View** creates an icon from a schematic (see Section 3−11−4 for more on this).
- **Make Schematic View** creates a schematic from a layout.
- **Make Alternate Layout View...** converts from layout or schematic to an alternate layout. You must choose a specific layout technology, and the new layout will use components from that technology.
- **Make Skeleton View** makes a skeletonized layout from a layout (the only thing in the skeleton is the exports and the frame; it is a "layout icon").
- **Make VHDL View** converts the current layout or schematic into structural VHDL. This VHDL is used by the Silicon Compiler (see Section 9−12) and the ALS simulator (see Section 9−5−2). Note that there are 5 schematic primitives which can exist in a normal and negated form ("buffer", "and", "or", "xor", and "mux"). You can choose the names to use for these two forms in the "Schematics" section of the "Technology" preferences (in menu **File / Preferences...**, "Technology" section, "Technology" tab).

3−11−3: Creating and Deleting Views

If the list of possible views is not sufficient to describe a cell, new views can be created with the **View Control...** command (in menu **View**). This command shows all views and lets you create and delete them.

When creating a new view, a name and an abbreviation are required. The abbreviation should be the first few letters of the full view name. This abbreviation will be used when describing cells with that view. For example, the view "fast−layout" might have the abbreviation "fast".


View Control

Views:

schematic {sch}
icon {ic}
layout {lay}
layout.skeleton {lay.sk}
layout.compensated {lay.comp}
VHDL {vhdl}
Verilog {ver}
documentation {doc}
documentation.waveform {doc.wave}
netlist {net}
netlist.netlisp {net.netlisp}

View name: fast-layout

Abbreviation: fast

Text view

The "Text View" checkbox indicates that this is a text−only view, like "Documentation", "Netlist", "Verilog", and "VHDL".

To delete a view, use the "Delete" button. You can delete only the views that you have created, not the basic views that exist on startup (such as "layout", "schematic", etc). Also, there must be no cells with the view that is being deleted.

3−11−4: Automatic Icon Generation

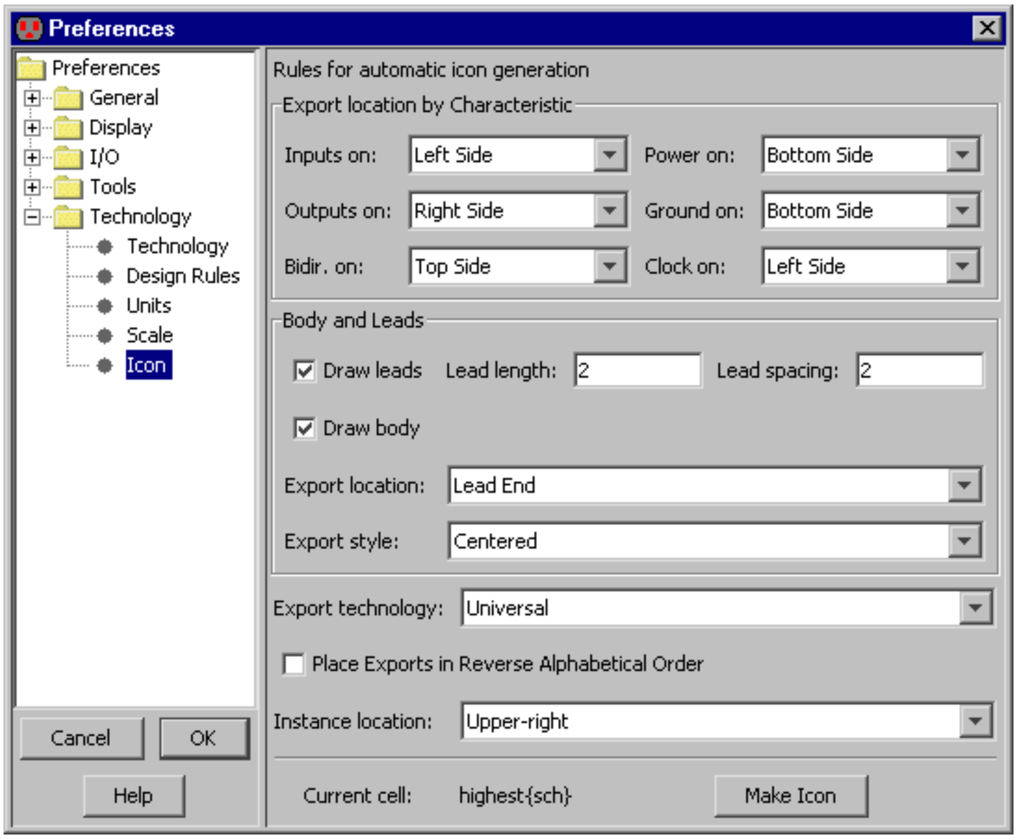
A particularly useful view type is icon. The icon cell is used for instances of an associated *contents* cell, which contains schematics. For example, you may have a cell called "adder{sch}" which contains a schematic. You may then create a cell called "adder{ic}" that contains a circle with a plus sign inside (these are nodes in the Artwork technology). This is then the icon for the contents cell "adder{sch}". Now, if you create an instance of the schematic cell, the icon cell will actually be placed, because it is the symbol that gets used for instances.

To generate an icon cell automatically, use the **Make Icon View** command (in menu **View**). Be sure to create all relevant exports before issuing this command, so that the proper icon can be constructed. Note that any export that has its "Body only" attribute checked will be omitted from the icon.

To control the look of the icons, use the "Icon" preferences (in menu **File / Preferences...**, "Technology" section, "Icon" tab). The top part of the dialog lets you place each of the different export types on any of the four sides of the icon.

The middle section of the dialog controls the body and leads of the icon.

You can choose whether or not to draw the body and leads. You can set the spacing and length of leads. You can choose the location of the exports (at the end of the leads, in the middle of the leads, or on the body). You can choose the style of the export text (whether it grows inward, outward).



Rules for automatic icon generation

Export location by Characteristic

Inputs on:	Left Side	Power on:	Bottom Side
Outputs on:	Right Side	Ground on:	Bottom Side
Bidir. on:	Top Side	Clock on:	Left Side

Body and Leads

[✓] Draw leads Lead length: 2 Lead spacing: 2

[✓] Draw body

Export location: Lead End

Export style: Centered

Export technology: Universal

[ ] Place Exports in Reverse Alphabetical Order

Instance location: Upper-right

Current cell: highest{sch}

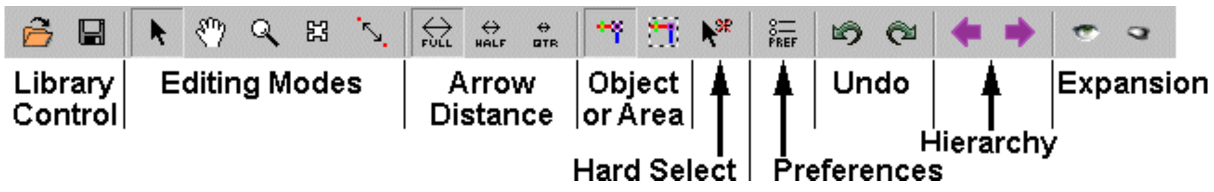
The bottom part of the dialog has miscellaneous controls. You can choose the technology of the exports ("Universal" uses nodes from the Generic technology which can connect to any arc; "Schematic" uses nodes from the Schematic technology and can connect only to other Schematic arcs). Exports are arranged alphabetically around the icon, and you can choose to reverse the alphabetical order. Finally, you can choose the location of the icon instance in the original schematic (when you use the **Make Icon View** command, it generates the icon and places an instance of that icon in the schematic). A button at the bottom requests that an icon be made now, and takes the place of the **Make Icon View** command.

The icon cell is correctly tied to its contents in most respects. If you descend into it (with the **Down Hierarchy** command in menu **Cell**), then you actually find yourself editing the associated contents cell. The **Up Hierarchy** command properly returns you to the location of the icon instance. Also, the network consistency checker and the simulators correctly substitute the contents whenever an icon appears. In order for this to work, however, all exports in the contents cell must exist with the same name in the icon cell (with the exception of those that are marked "Body Only").

Chapter 4: Display

4−1: The Tool Bar

The tool bar sits near the top of the screen, below the menu bar. It provides shortcuts for many common commands.



The tool bar has these sections:

- **Library Control** Icons to read a library (Section 3−9−2) and to save libraries (Section 3−9−3).
- **Editing Modes** Icons for selection (Section 2−1−1), panning (Section 4−4−2), zooming (Section 4−4−1), outline edit (Section 6−10−2), and measuring (Section 4−7−4).
- **Arrow Distance** Icons set the distance that arrow keys will move, to full, half, and quarter units (Section 2−4−1).
- **Object or Area** Icons switch between object selection and area definition (Section 2−1−3).
- **Hard Select** Icon to toggle the selection of hard−to−select objects (Section 2−1−5).
- **Preferences** Icon to show the preferences dialog (Section 6−3).
- **Undo** Icons to undo and redo (Section 6−7).
- **Hierarchy** Icons to go back and forward while traversing the hierarchy (Section 3−5).
- **Expansion** Icons to expand and unexpand cell instances (Section 3−4).

4−2: The Messages Window

The messages window is a text window near the bottom of the screen. Many commands list their results in the messages window, and minor error messages are reported there.

The text in the messages window can be selected with the cursor and edited with the **Cut**, **Copy**, and **Paste** commands (in menu **Edit**). You can remove all text with the **Clear** command (in menu **Window / Messages Window**). In addition, you can right−click in the messages window to Cut, Copy, Clear, or Paste the selected text. You can also use the right−click context menu to Cut or Copy all of the text in the window.

The text in the messages window can be saved to disk by using the **Save Messages...** command (in menu **Window / Messages Window**). You will be prompted for the place to save the text. You can select the messages window font with the **Set Font...** command.

Chapter 4: Display

4−3: Creating and Deleting Editing Windows

Initially, there is only one editing window on the screen. Electric allows you to create multiple editing windows, each of which can show a different cell. You can also have the same cell in more than one window to see it at different scales and locations.

New windows are created by checking the appropriate checkbox in the **New Cell...** or **Edit Cell...** commands (in menu **Cell**). New windows can also be created from the cell explorer by using the context button on a cell name.

To delete a window, click its close box, or use the **Close Window** command (in menu **Window**). Note that you cannot delete the last window on UNIX systems, because the UNIX pulldown menus are part of the edit windows.

When there are many editing windows on the display, you can arrange them neatly with the **Window / Adjust Position** commands. The **Tile Horizontally** command adjusts the windows so that they are full−width, but just tall enough to fill the screen, one above the other. The **Tile Vertically** command adjusts the windows so that they are full−height, but just wide enough to fill the screen, one next to the other. The **Cascade** command adjusts the windows so that they are all the same size and overlap each other uniformly from the upper−left to the lower−right.

4−4−1: Scaling

The scale of a window's contents can be controlled in a number of ways. The **Zoom In** command (in menu **Window**) zooms in, magnifying the contents of the display. The **Zoom Out** command does the opposite − it shrinks the display. Both zoom by a factor of two.

During normal editing, you can zoom the display with the shift−right button (see Section 1−8). Holding shift−right while dragging a rectangular area causes the display to zoom into that area, making it fill the screen. Clicking shift−right in a single location causes the display to zoom out, centered at that point.

You can also use the Zoom tool from the tool bar to zoom in and out. This has the same zoom in and out functions, but they are now attached to the left button (no shift needed). To zoom into an area, click and drag out that area. To zoom out, hold the shift key and click in the center of the desired area. This mode can also be invoked with the **Toggle Zoom** command (in menu **Edit / Modes / Edit**).

The most useful scale change command is **Fill Window** (in menu **Window**), which makes the current cell fill the window.

To examine a specific area of the display, use the **Focus on Highlighted** command (in menu **Window / Special Zoom**), which makes the highlighted objects fill the display. To examine a specific area of the display that is not necessarily aligned with nodes and arcs, use the area select commands (see Section 2−1−3). You can also use the **Zoom Box** command, which allows you to drag−out a rectangle, and then zooms to that area.

The **Make Grid Just Visible** command (in menu **Window / Special Zoom**) zooms in or out until the grid is minimally visible. Any futher zoom−out from this point will make the grid invisible. If the grid is not being displayed, it is turned on. See Section 4−7−1 for more on the grid.

A final scaling command is **Match Other Window** which redraws the current window at the same scale as the other. If there are more than two windows, you will be asked to select the window to match.

4−4−2: Panning

Besides scaling, you can also pan the window contents, shifting it about on the display. This is typically done with the sliders on the right and bottom of the window. On systems that have a mouse wheel, you can use that to pan vertically (and hold the shift key while rolling the mouse wheel to pan horizontally).

You can also use the Pan tool from the tool bar to move the window contents. And this mode can be invoked with the **Toggle Pan** command (in menu **Edit / Modes / Edit**).

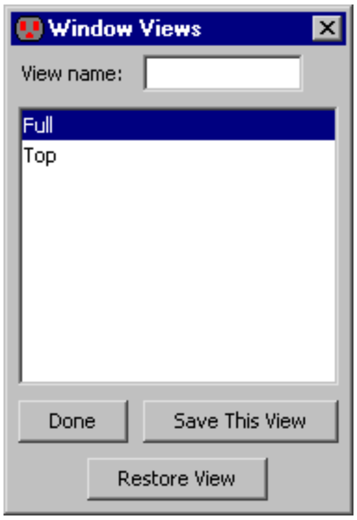
In addition to these methods, panning can also be done from menu commands. The **Pan Left**, **Pan Right**, **Pan Up**, and **Pan Down** commands (in menu **Window**) all shift the window contents appropriately (and because they are bound to quick keys, these operations can even be done from the keyboard). By default, these commands shift the screen by about 30% of its size. You can use the "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab), to change that amount. The **Small** panning distance causes subsequent shifts to be about 15% of the screen size. The **Medium** panning distance causes subsequent shifts to be about 30% of the screen size. The **Large** panning distance causes subsequent shifts to be about 60% of the screen size.

The **Center** commands (in menu **Window**) are rarely−used panning commands for shifting the window contents without scaling. There are two commands: **Selection** makes the window shift so that the highlighted objects are in the center of the window, and **Cursor** makes the window shift so that the current cursor location is in the center of the window. Note that this second command is useful only when bound to a keystroke, because you cannot issue the command and have a valid cursor location at the same time.

One final command is useful if the display appears incorrect. If this happens, redraw the screen with the **Redisplay Window** command (in menu **Window**).

4−4−3: Saving Views

Once a particular scale and position is established in a window, you can save it and retrieve it later. The **Saved Views...** command (in menu **Windows**) presents a dialog that lists saved views. You can name the current view and save it with the "Save This View" button. A previously saved view can be displayed with the "Restore View" button.



4−5: Layer Visibility

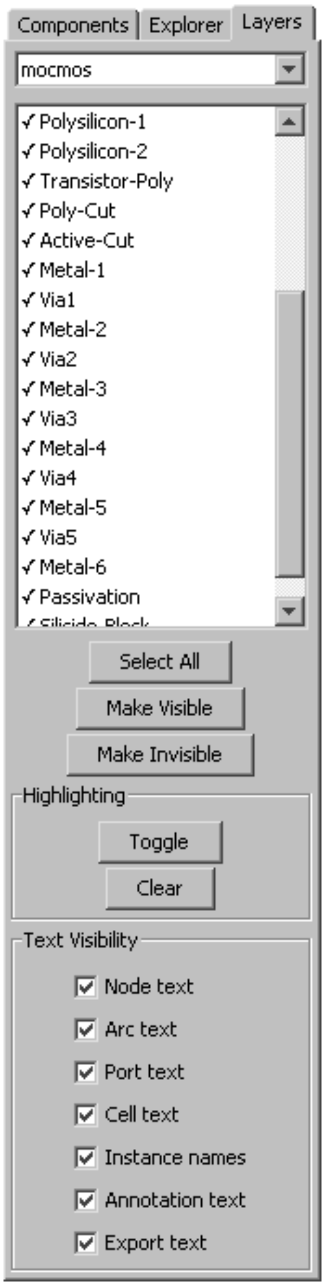
The nodes and arcs on the display are composed of more basic *layers*. By using the "Layers" tab of the Side Bar, you can control which layers are actually drawn.

The layers tab shows the layers in the current technology. Changing the technology popup at the top of this tab will change the current technology. When a layer is checked, it is visible. You can turn the check on and off by double−clicking on a line. You can also use the "Make Visible" and "Make Invisible" buttons. The "Select All" button selects every layer so that the "Make..." buttons will work on the entire set.

Note that the layers are listed in order of height, and that you can select multiple entries in the list by using the Shift key. This means that you can easily control visibility by depth in the chip.

Two buttons in the middle control the *highlighting* of layers. By selecting a layer and clicking "Toggle", it makes that layer stand out on the display. Use "Clear" to return to normal layer display.

The bottom of the tab lets you choose which of the different types of text will be visible. These different types of text are described more fully in Section 6−8−1.



mocmos

✓ Polysilicon-1
✓ Polysilicon-2
✓ Transistor-Poly
✓ Poly-Cut
✓ Active-Cut
✓ Metal-1
✓ Via1
✓ Metal-2
✓ Via2
✓ Metal-3
✓ Via3
✓ Metal-4
✓ Via4
✓ Metal-5
✓ Via5
✓ Metal-6
✓ Passivation

[✓] Node text
[✓] Arc text
[✓] Port text
[✓] Cell text
[✓] Instance names
[✓] Annotation text
[✓] Export text

4−6−1: Electric's Color Model

There are two preferences panels that control the appearance of individual layers in the editing window. These are the "Colors" and "Layers" preferences (in menu **File / Preferences...**, "Display" section). Before explaining these commands, it is useful to understand the distinction between *transparent* and *opaque* layers in Electric.

Every layer in a technology is either transparent or opaque. Transparent layers are able to overlap each other, and it is possible to see all of them. Typically, the most commonly used layers are transparent because it is clearer to distinguish.

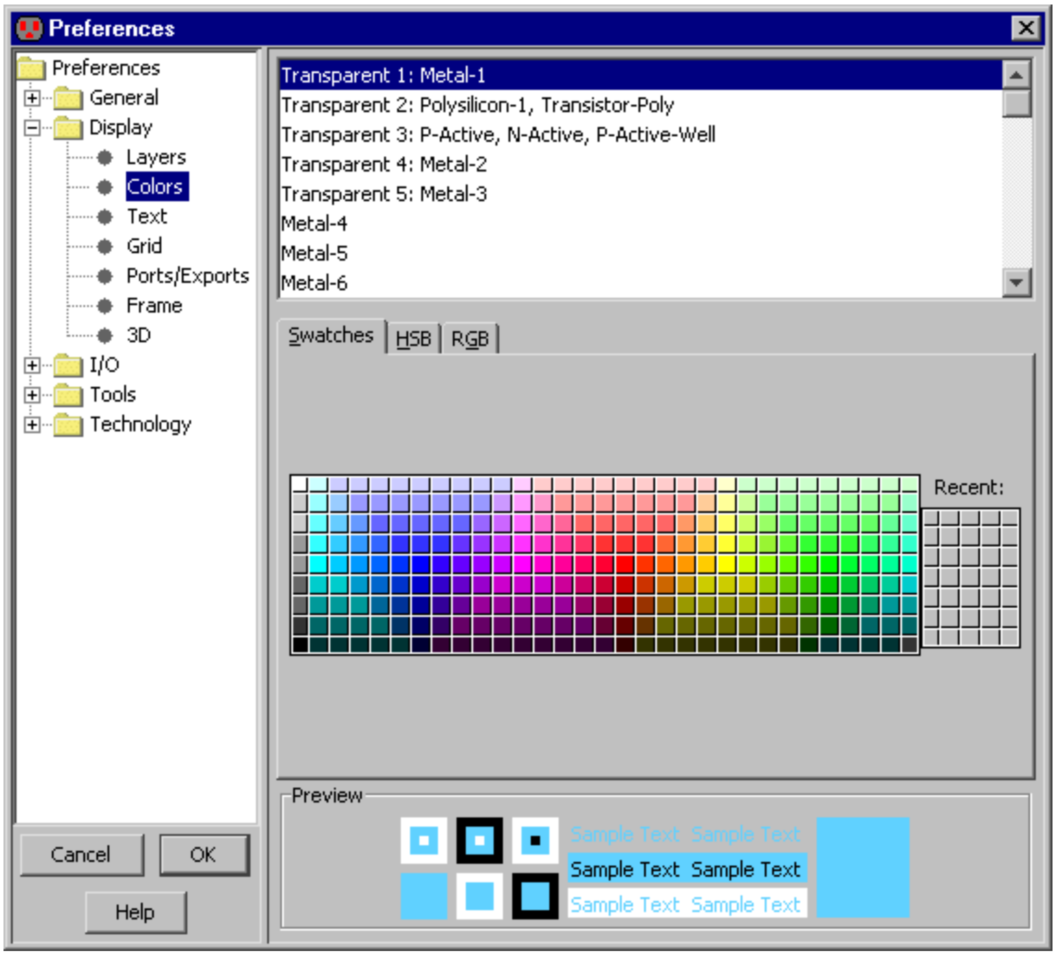
The remaining layers in a technology are opaque, meaning that when drawn, they completely obscure anything underneath. These layers typically have stipple patterns so that they do not cover all of the bits. In this way, the opaque layers can combine without obscuring the display. Because opaque color does obscure everything under it, the less common layers are drawn in this style.

When editing colors, the opaque layers have only one color, whereas the transparent layers have many different colors, considering their interaction with other transparent layers.

4−6−2: Editing Colors

The first color control is the "Colors" preferences (in menu **File / Preferences...**, "Display" section, "Colors" tab). The top half of the dialog lists all possible things that can have their color changed:

- The transparent colors (all layers drawn transparently are listed).
- The opaque layers.
- Special colors such as the background, text, waveform windows, etc.



Transparent 1: Metal-1
Transparent 2: Polysilicon-1, Transistor-Poly
Transparent 3: P-Active, N-Active, P-Active-Well
Transparent 4: Metal-2
Transparent 5: Metal-3
Metal-4
Metal-5
Metal-6

When changing the background color, note that it must contrast with both the highlight color and the inverse of the highlight color (the inverse is black in the default settings).

Electric has three different color schemes which have predefined colors. The commands of the **Window / Color Schemes** menu let you switch between them. With the **Black Background Colors** command, the background becomes black; with the **White Background Colors** command, the background becomes white; with the **Restore Default Colors** command, the background becomes gray (the default).

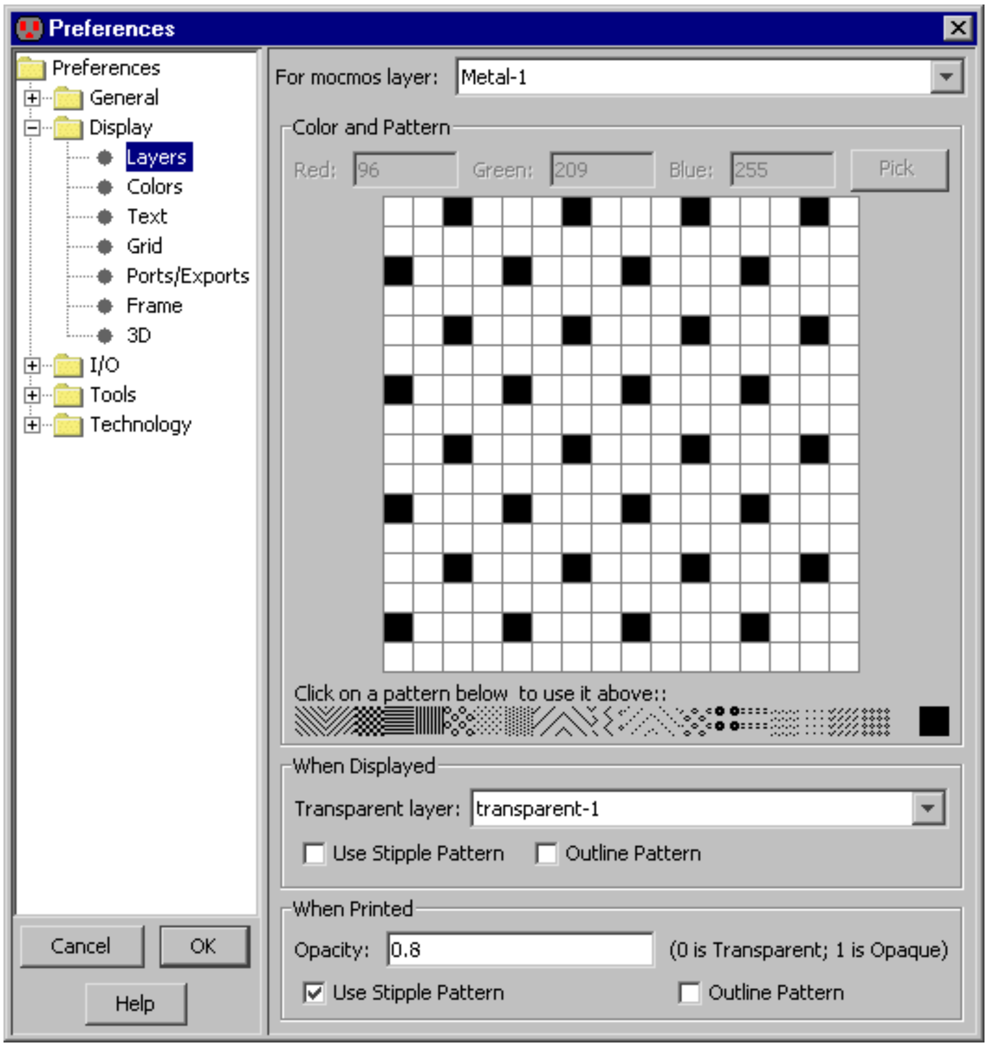
4−6−3: Editing Patterns

The "Layers" preferences (in menu **File / Preferences...**, "Display" section, "Layers" tab) lets you control the appearance of individual layers, including their stipple patterns.

Each layer has a color and pattern in the top section. You can draw in the pattern area to set a pattern, and you can choose from a set of predefined patterns by clicking on their image below the pattern−editing area.

A layer's color is only editable if it is opaque. When transparent, you must use the "Colors" tab to set the color.

Two sections at the bottom control the appearance of the layer on the screen and on the printed page. The display section lets you assign the layer to one of the transparent layers. You can also choose whether to draw the layer in solid colors or use the specified pattern (and if patterned, you can choose to outline the pattern or not).



For mocmos layer: Metal-1

Color and Pattern

Red: 96 Green: 209 Blue: 255

Click on a pattern below to use it above::

When Displayed

Transparent layer: transparent-1

☐ Use Stipple Pattern ☐ Outline Pattern

When Printed

Opacity: 0.8 (0 is Transparent; 1 is Opaque)

☑ Use Stipple Pattern ☐ Outline Pattern

The printing section also lets you choose patterned vs. solid and whether to outline the pattern. In addition, it lets you set an opacity which will control its appearance on the printed page.

4−7−1: Drawing a Grid

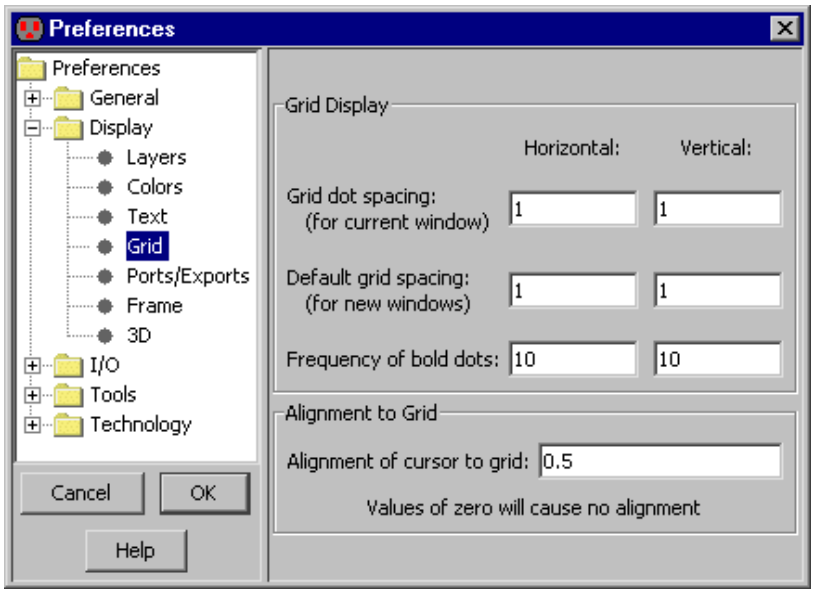
The **Toggle Grid** command (in menu **Window**) turns the grid display on and off. The grid consists of dots at every grid unit, and bolder dots every 10 units, but both of these distances are settable.

Initially, the grid dots are spaced 1 unit apart. The size of a grid unit can be related to real−world distance by considering the *scale* of the technology. For example, in the MOSIS CMOS technology, the scale is 0.2 microns, as shown in the status area. When the grid is displayed, the dots are therefore 0.2 microns apart. For more information on scaling, Section 7−2−1.

Note that the grid display changes as you zoom in and out. When zoomed too far out to show all of the dots, only the bolder dots are shown. When zoomed too far out to show even the bolder dots, the grid is not displayed. However, the fact that the grid should be on is remembered, so it reappears when you zoom back in.

The "Grid" preferences (in menu **File / Preferences...**, "Display" section, "Grid" tab) presents a dialog in which grid spacing may be set. You can change the grid spacing for the current window, and also set a default grid spacing to be used in new windows.

It is possible to change the horizontal and vertical grid dot spacings. You can also change number of grid dots between bold ones.



Grid Display

	Horizontal:	Vertical:
Grid dot spacing: (for current window)	1	1
Default grid spacing: (for new windows)	1	1
Frequency of bold dots:	10	10

Alignment to Grid

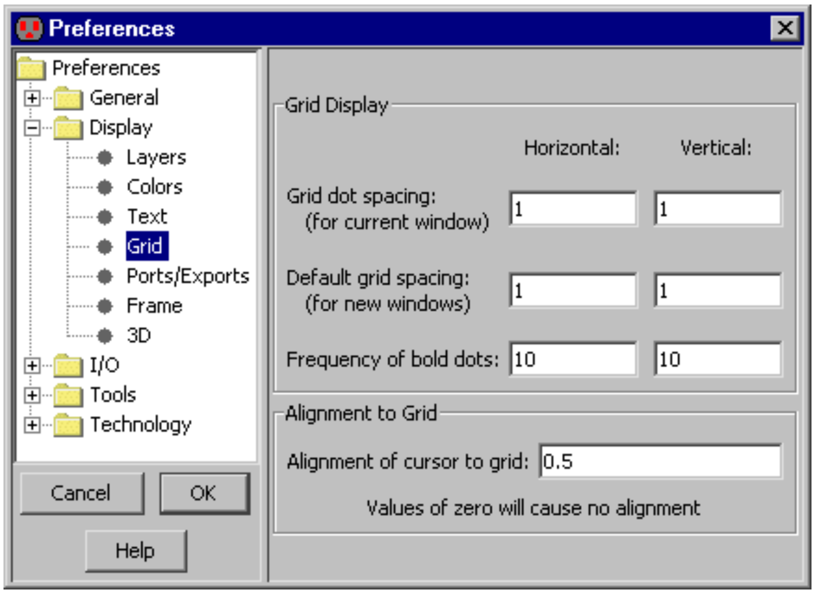
Alignment of cursor to grid: 0.5

Values of zero will cause no alignment

The grid spacing is used by arrow keys when they move objects (see Section 2−4−1 for more on arrow key motion).

4−7−2: Aligning to a Grid

When moving or creating circuitry, the cursor location is snapped to a grid so that editing is cleaner. This snapping is controlled by the alignment options (which are not necessarily the same as the grid options).



Grid Display

	Horizontal:	Vertical:
Grid dot spacing: (for current window)	1	1
Default grid spacing: (for new windows)	1	1
Frequency of bold dots:	10	10

Alignment to Grid

Alignment of cursor to grid: 0.5

Values of zero will cause no alignment

The "Grid" preferences (in menu **File / Preferences...**, "Display" section, "Grid" tab) presents a dialog in which alignment values may be set. For example, if the grid spacing is 2x3, and the alignment is 0.5, then there are up to six different positions for placement inside a displayed grid rectangle.

The **Align to Grid** command (in menu **Edit / Move**) cleans up the selected objects by moving them to aligned coordinates. This is useful for circuitry that has been imported from external sources, and needs to be placed cleanly for further editing.

4−7−3: Aligning to Objects

It is often the case that a collection of objects should line−up uniformly. The commands of the **Edit / Move** menu offer six possible ways to do this.

The command **Align Horizontally to Left** (and **Align Horizontally to Right**) moves all of the objects so that their left edge (or right edge) is moved to the leftmost (or rightmost) location of those objects. The command **Align Horizontally to Center** moves all of the objects so that their X center is at the location of the X center coordinate of those objects.

The command **Align Vertically to Top** (and **Align Vertically to Bottom**) moves all of the objects so that their top edge (or bottom edge) is moved to the topmost (or bottommost) location of those objects. The command **Align Vertically to Center** moves all of the objects so that their Y center is at the location of the Y center coordinate of those objects.

4−7−4: Measuring

If you wish to find the distance between any two points on the display, use the "Measure" tool from the tool bar.

This mode can also be invoked with the **Toggle Measure Distance** command (in menu **Edit / Modes / Edit**). Another way to measure distances is to use the cursor coordinates, displayed in the status area.

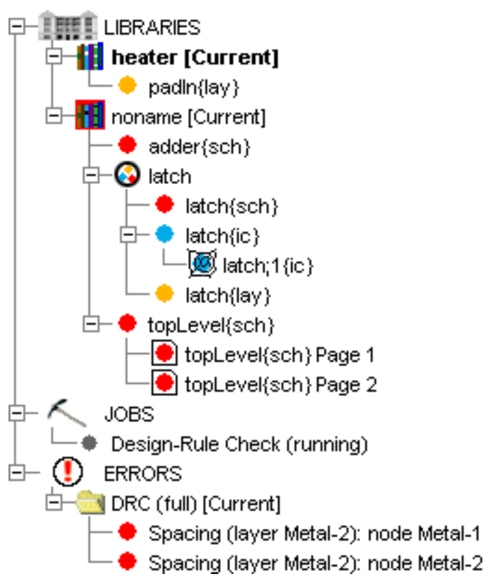
In measure mode, each click places a new point on the display, and shows the distance to the previous point. Clicking the right button lets you start a new measure point without connecting it to the previous one. Double−clicking the right button removes the measurements.

The measured distance can be used by the **Array...** command (in menu **Edit**) to specify spacing (see Section 6−4).

4−8: The Cell Explorer

The side bar sits on the left side of every window. You can move it to the right side with the **On Right** command (of menu **Windows / Side Bar**) and move it back with the **On Left** command. You can also request that the side bar always be on the right by checking "Side bar defaults to the right side" in the "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab)

The cell explorer resides in the "Explorer" tab of the side bar. It shows a hierarchical tree with three main sections: LIBRARIES, JOBS, and ERRORS.



The LIBRARIES section of the explorer lists all libraries and cells. You can examine them in three different ways:

- **Alphabetically** all cells are listed alphabetically.
- **By group** all cells are listed alphabetically, but are also organized into cell groups.
- **By hierarchy** only the "top level" cells of each library are listed (top level cells are those that are not used as instances in any other cells). Inside of a cell are the subcells that comprise it, along with the number of times that that cell appears.

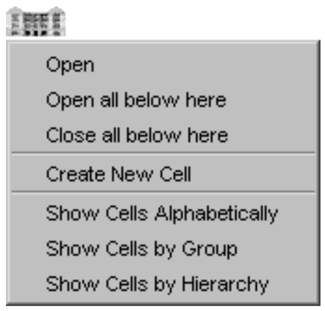
To change the view, right−click on the LIBRARIES icon and choose a view. Note that libraries which have been modified are listed in bold−face.

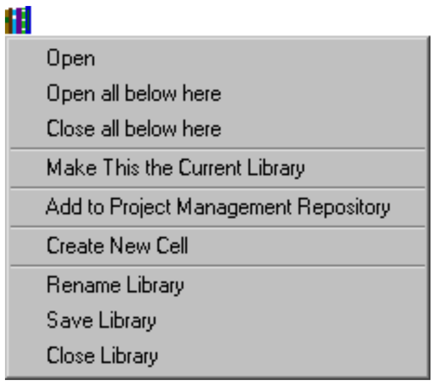
The second section of the explorer is the JOBS section. Here are listed all running tasks in Electric. The section is usually empty, but if multiple jobs are running at the same time, you can examine them and manipulate them.

The third part of the cell explorer is the ERRORS section. This lists all errors that were generated by other tools (DRC, ERC, NCC, etc.) and which can be examined with the "<" and ">" keys.

Many special functions can be done in the cell explorer. You can double−click on any cell name to see that cell in the right half of the window. There are special context menus available by right−clicking on an entry (use command−click on the Macintosh).

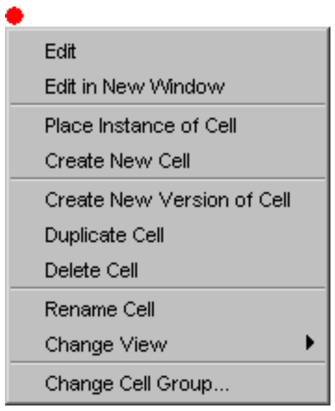
The context menu for the LIBRARIES icon has 3 parts. The top three entries let you control the expansion of the tree. The middle entry lets you create a new cell. The bottom three entries lets you view the libraries in different ways (explained above).

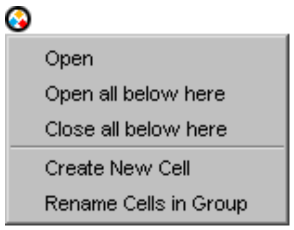




The context menu for each library icon has 5 parts. The top three entries let you control the expansion of the tree. The next entry lets you make the library the current library. The next entry lets you manage the library with Project Management (see Section 6−12). The next entry lets you create a new cell in the library. The bottom three entries let you rename, save or delete the library.

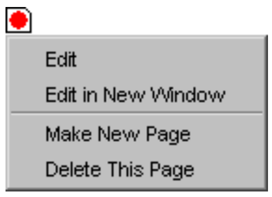
The context menu for each cell icon has 5 parts. The top two entries let you edit the cell (in the current or in a new window). The next two entries let you place an instance of the cell and create a new cell. The next three entries let you create a new cell version, create a new cell copy, or delete the cell. The next two entries let you rename the cell or change its view. The bottom entry lets you rearrange cell groups.



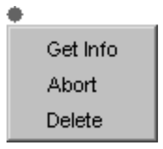


The context menu for each cell group has 2 parts. The top three entries let you control the expansion of the tree. The bottom entries let you create a new cell in the group, or rename every cell in the group.

The context menu for a multi−page schematic cell has two parts (see Section 7−5−2 for more on multi−page schematics). The top two entries let you edit the cell (in the current or in a new window). The bottom entries let you add a new page to the current multi−page schematic, or delete the current page of the multi−page

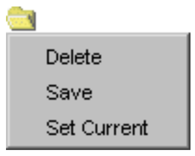


schematic.



The context menu for individual jobs under the JOBS icon has 3 entries: "Get Info" requests any additional information about the job; "Abort" requests that the Job stop itself (not always possible); and "Delete" removes a job from the queue.

The context menu for each collection of errors in the ERRORS section has 3 entries: "Delete" removes this set of errors; "Save" saves the errors to a disk file; and "Set Current" makes this the current set of errors (which can be examined with the "<" and ">" keys).



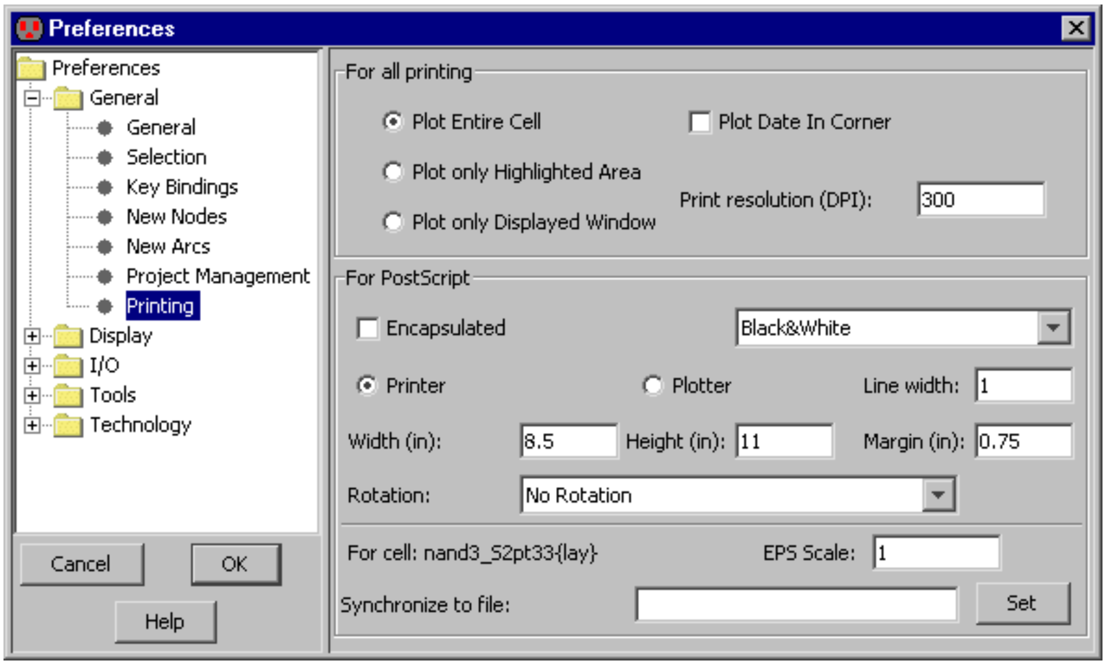
4−9: Printing

To make a paper copy of the contents of the current window, use the **Print...** command (in menu **File**). You can use the **Page Setup...** command for general print settings.

As an alternative to printing, you can request the system to write a PostScript, HPGL, or PNG file. To do this, use the **PostScript...**, **HPGL**, and **PNG (Portable Network Graphics)...** commands (in menu **File / Export**).

For specific printing and PostScript settings, use the "Printing" preferences (in menu **File / Preferences...**, "General" section, "Printing" tab).

The "For all printing" section at the top has some general options. The default is to include the entire cell, but you can choose to print only what is highlighted or only what is displayed by selecting the appropriate buttons.


Preferences

For all printing: [✓ Plot Entire Cell] Plot only Highlighted Area / Plot only Displayed Window; Plot Date In Corner: ☐; Print resolution (DPI): 300

For PostScript: Encapsulated: ☐; Black&White; [✓ Printer] Plotter; Line width: 1; Width (in): 8.5; Height (in): 11; Margin (in): 0.75; Rotation: No Rotation

For cell: nand3_S2pt33{lay}; EPS Scale: 1; Synchronize to file:

Note that when printing the highlighted area, a precise selection can be made with Area selection (see Section 2−1−3).

The "Plot Date In Corner" option causes additional information to appear in the corner of the plot.

The "Print resolution" is the number of dots−per−inch (DPI) that the printer expects. Higher resolutions use more memory for the print image.

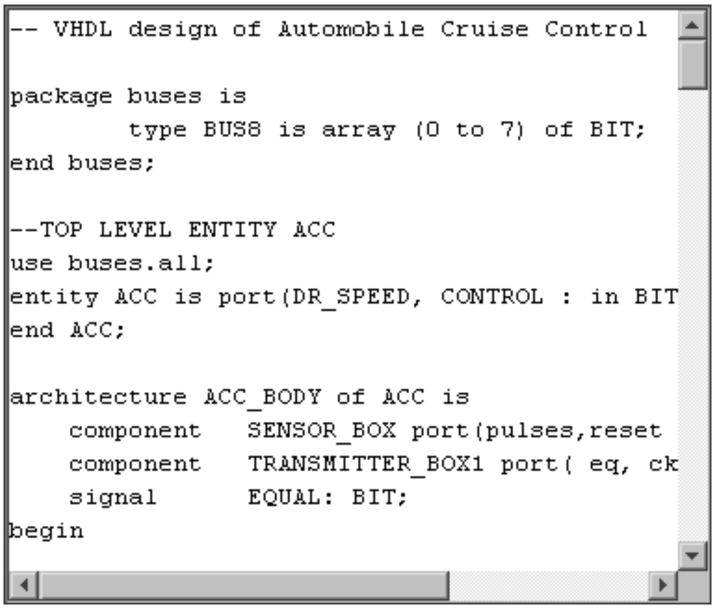
There are many PostScript options, available in the lower section.

- The "Encapsulated" checkbox causes the PostScript output to be insertable in other documents.
- There are three color choices: "Black&White", which uses stipple patterns for the layers; "Color" which uses solid colors, but does not handle overlap (because PostScript does not handle transparency); and "Color Stippled" which uses color stipple patterns for better overlap.
- You can specify the size of the page (choose "Printer" for devices that print onto single pieces of paper, and "Plotter" for devices that print onto continuous rolls of paper). The "Margin" field is the amount of white space to leave on the sides. All distances in the "Height", "Width", and "Margin" fields are in inches.
- You can control the width of PostScript lines. Although they default to 1, this may be too thin on some printers.
- You can choose to rotate the image by 90 degrees so that it fits better on the page. The default is "No Rotation", but the popup can switch to "Rotate plot 90 degrees" or "Auto−rotate plot to fit".
- You can request that PostScript files be synchronized with the current cell. Clicking the "Set" button prompts you for a file name, which is stored with the current cell. Whenever you write any PostScript, Electric checks all synchronized cells to see if they are newer than their associated disk file. If they are newer, the files are regenerated. Thus, you can specify PostScript files for many different cells in a library, and when PostScript is generated, all of the files will be properly updated to reflect the state of the design.

4−10: Text Windows

Some cells are textual in nature (VHDL, Verilog, Netlists, or Documentation), and cause text to appear in the edit window.

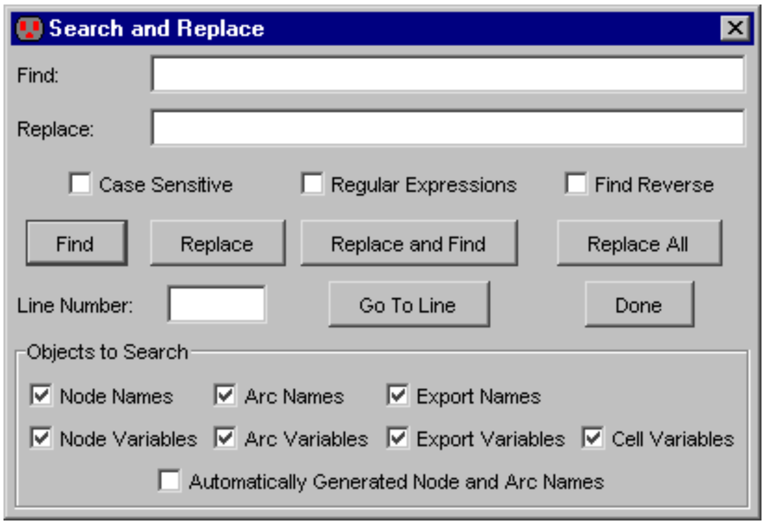
When text editing, the standard point−and−click commands apply. You can use the **Cut**, **Copy**, and **Paste** commands (in menu **Edit**).



```
-- VHDL design of Automobile Cruise Control

package buses is
        type BUS8 is array (0 to 7) of BIT;
end buses;

--TOP LEVEL ENTITY ACC
use buses.all;
entity ACC is port(DR_SPEED, CONTROL : in BIT
end ACC;

architecture ACC_BODY of ACC is
    component   SENSOR_BOX port(pulses,reset
    component   TRANSMITTER_BOX1 port( eq, ck
    signal      EQUAL: BIT;
begin
```

The contents of a text window can be saved to disk with the **Text Cell Contents...** command (in menu **File / Export**) and restored from disk with the **Text Cell Contents...** command (in menu **File / Import**).

Note that there is no "saving" of text windows because they are editing internal data structures. Therefore every change updates the information in Electric (but the library must be saved to truly preserve changes).

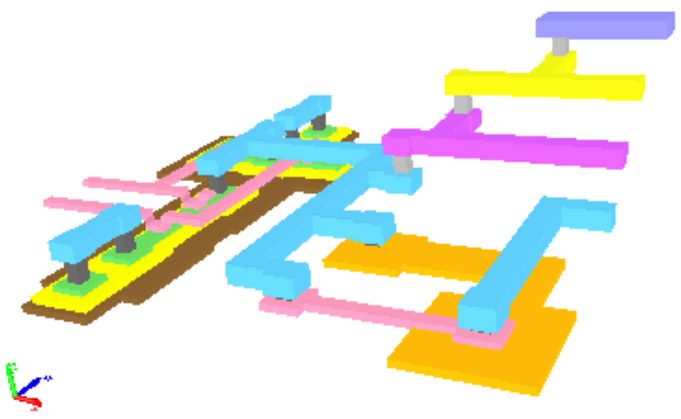


Searching is done with the **Find Text...** command (in menu **Edit / Text**). You can find and/or replace text with the appropriate buttons. Check boxes allow the search to be case sensitive, have regular expressions, and to go in the reverse direction. In addition, you can jump directly to a specified line number.

Interestingly, the **Find Text...** command can also be used outside of the text edit window. If you are editing a layout or schematic, this dialog will search all of the node, arc, export, and other names. The checkboxes in the "Objects to Search" area control which of these pieces of text will be considered. "Automatically Generated" names are those created for you by the system. They can be included in the search but normally are not.

4−11−1: Introduction

Electric has the ability to view an integrated circuit in 3−dimensions as shown below, allowing a fuller understanding of the interaction between layers. When displaying 3D, you can rotate, zoom, and pan the image to get a better view, however you can no longer change the circuit.



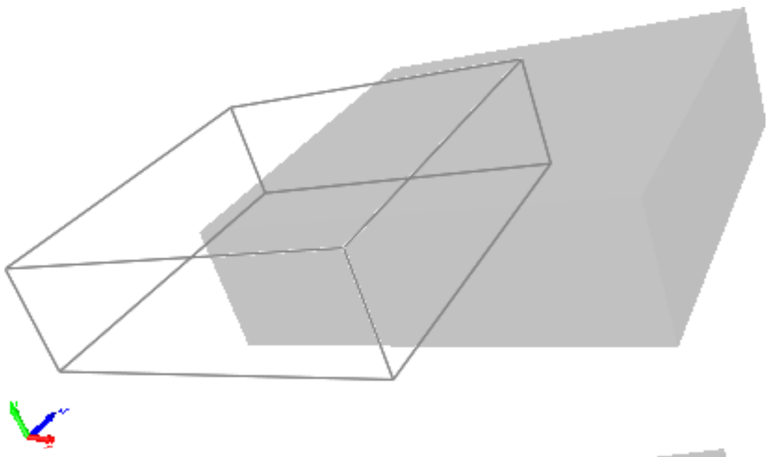
The 3D View is based on Java3D, the Java interface for interactive 3D graphics. Because not everyone has a full 3D capability on their computer, the 3D facilities are dependent on these extra installations:

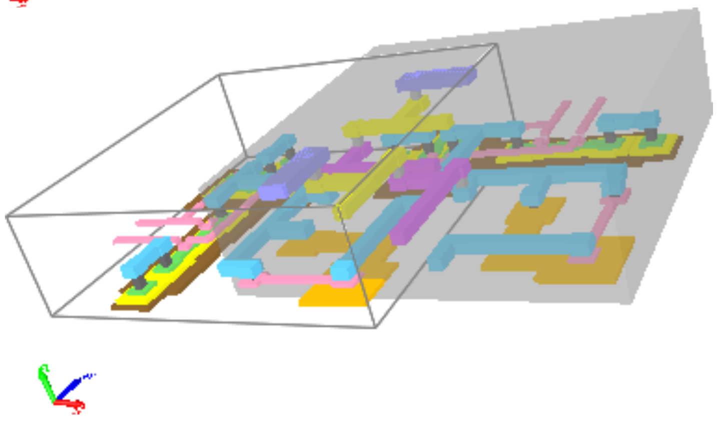
- **Java3D** is the core 3D package and must be installed.
- **3D axes** is an optional extra download from Static Free Software that shows a 3D axis.
- **JMF** is an optional package from Sun Microsystems that enables animation.
- **Animation** is an optional extra download from Static Free Software that does animation (it needs JMF).

See Section 1−5 for details about getting these extensions.

To see the 3D view of a layout cell, use the **3D View** command (in menu **Window / 3D Window**). The cell is displayed in 3D, and mouse movements will rotate, pan, or zoom the circuit. Use the left button to rotate, the right button for panning, and the middle one for zooming. When zooming, drag the middle button in one direction to zoom in, and the other direction to zoom out. Standard pan and zoom operations (in menu **Window**) are also available (see Section 4−4−1 and Section 4−4−2).

Each layer of a node or arc is drawn as a separate object in the 3D view. If you click on a node or arc in a 2D view, all of its layers will be highlighted in the 3D view. Conversely, clicking on any layer of a node or arc in the 3D view will show the entire component in the 2D view.





Cell instances will be drawn as bounding boxes if they are unexpanded (top illustration), and will show their contents if expanded (bottom illustration).

Troubleshooting

If you are running on Windows and are using MDI mode (multiple document interface) the 3D display may not work properly. See Section 1−4 for instructions on running Electric in SDI mode.

Because Java3D makes use of the graphics hardware on your computer, it may be useful to test that hardware with the **Test Hardware** command (in menu **Window / 3D Window**).

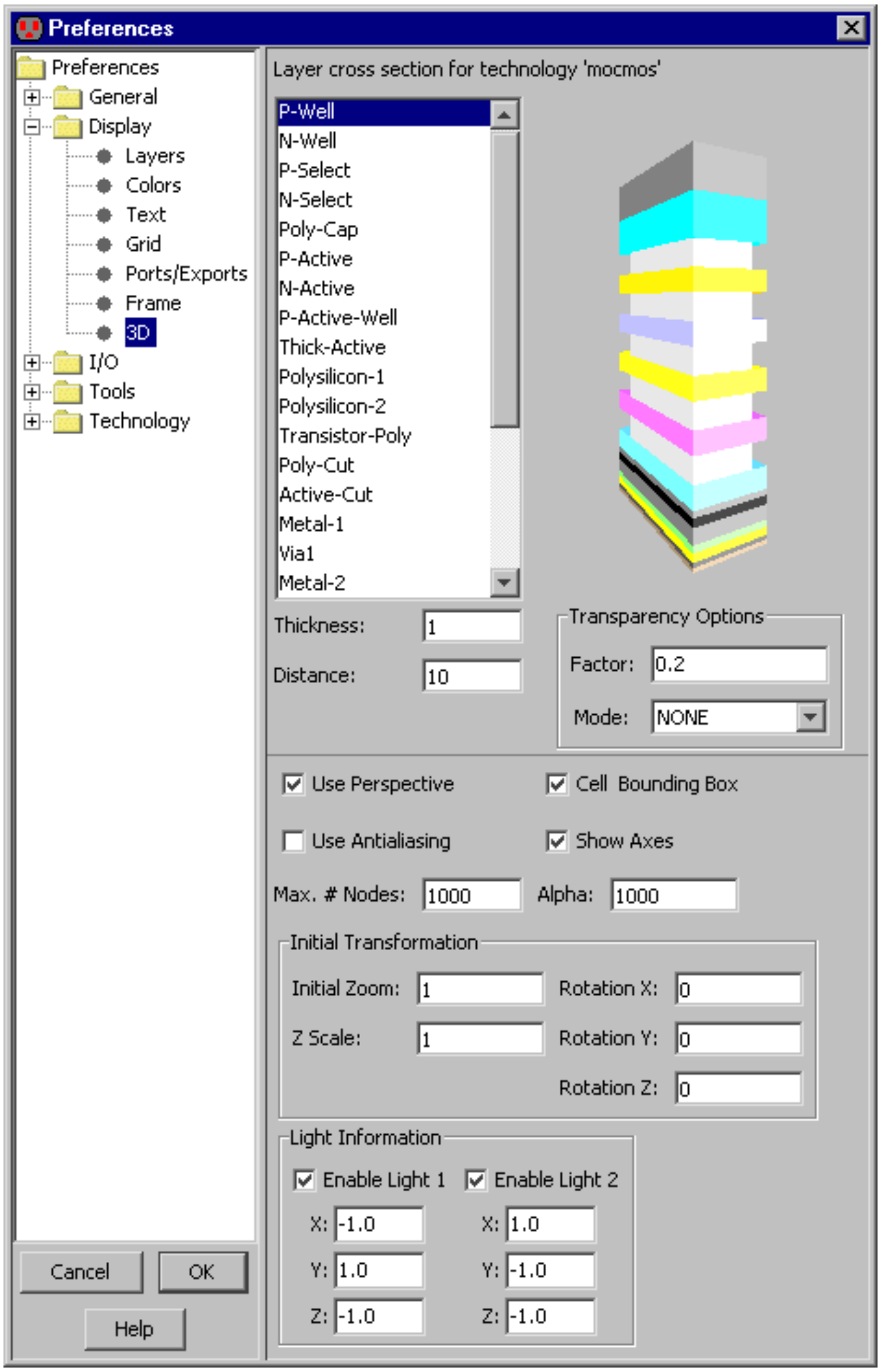
4−11−2: 3D Preferences

To control the 3D view, use the "3D" preferences (in menu **File / Preferences...**, "Display" section, "3D" tab). This provides access to most of the parameters that control 3D viewing. The only other controls available are the colors used to draw 3D features, which are available in the "Colors" preferences (see Section 4−6−2).

In the 3D preferences, the thickness and Z distance (height) of each layer can be controlled as well as the view mode, the Z−axis scale, and use of antialiasing.

On the left side of this dialog is a list of layers in the current technology. On the right side is a cross sectional view of the chip, showing the relative position of each layer. You can select a layer by clicking on either side of the dialog. The currently selected layer is highlighted in the list on the left and shown in red in the right−hand view. Change the "3D HIGHLIGHTED INSTANCES" entry in the "Color" preferences to change the color used for highlighting layers in the 3D view and in the preferences.

The distance of the layer from the wafer bottom and its thickness are the most important values. These values are not only used for the 3D view; they are also used whenever layers are presented in "height" order. Once selected, you can type new values into the "Thickness" and "Distance" fields.



Layer cross section for technology 'mocmos'

P-Well
N-Well
P-Select
N-Select
Poly-Cap
P-Active
N-Active
P-Active-Well
Thick-Active
Polysilicon-1
Polysilicon-2
Transistor-Poly
Poly-Cut
Active-Cut
Metal-1
Via1
Metal-2

Thickness: 1
Distance: 10

Transparency Options
Factor: 0.2
Mode: NONE

☑ Use Perspective ☑ Cell Bounding Box
☐ Use Antialiasing ☑ Show Axes

Max. # Nodes: 1000 Alpha: 1000

Initial Transformation
Initial Zoom: 1 Rotation X: 0
Z Scale: 1 Rotation Y: 0
Rotation Z: 0

Light Information
☑ Enable Light 1 ☑ Enable Light 2
X: -1.0 X: 1.0
Y: 1.0 Y: -1.0
Z: -1.0 Z: -1.0

By default, a perspective view is shown. Uncheck "Use Perspective" to see a parallel display. Antialiasing can be turned on by checking "Use Antialiasing". Due to performance issues, antialiasing is not on by default.

The transparency option controls whether you can see through layers, allowing finer control of the display. The transparency factor ranges from 0 (fully opaque: not transparent at all) to 1 (completely transparent: an invisible shape). The transparency mode sets the rasterization technique to use during rendering. Possible values are NONE, BLENDED, FASTEST, NICEST or SCREEN DOOR. The default setting of "NONE" indicates that all objects are opaque. Refer to www.j3d.org for technical details.

Other controls are available in this dialog, for example the initial zoom factor and rotation. If the displayed layers are too thin along the Z axis (compared to their X and Y values), use the "Z Scale" field to make everything thicker.

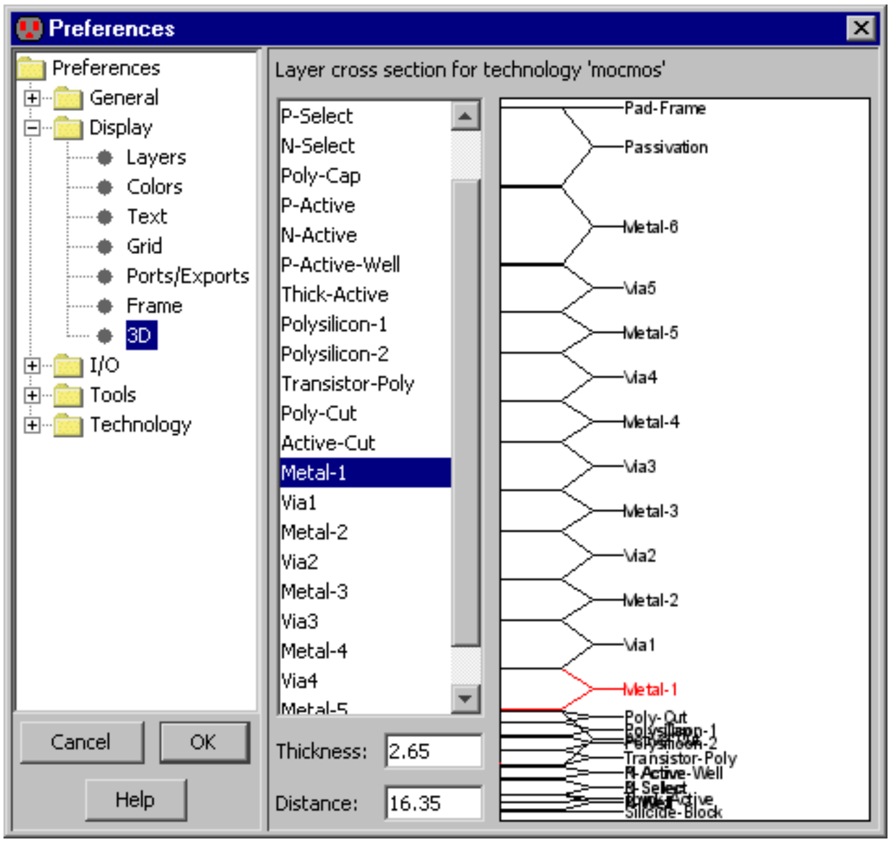
Lights

The 3D view uses one the ambient (background) light and two directional lights. The ambient light is always on, but the directional light can be enabled or disabled with the checkboxes.

The directional lights sit outside of the circuit and point in the given direction. The default directions of (−1, 1, −1) and (1, −1, −1) illuminate the 3D view from the front. Although the lights have a default color of white, this can be changed by editing the "SPECIAL: 3D DIRECTIONAL LIGHT" entry in the "Color" preferences.

Ambient light is the background light that fills a room. It is used to illuminate those areas that are not directly hit by the directional lights. The default color of the ambient light is gray, but this can be changed by editing the "SPECIAL: 3D AMBIENT LIGHT" entry in the "Color" preferences.

If Java3D is not installed, the distance and the thickness can still be controlled. In such a situation, the 3D preferences dialog has much more limited information.



Thickness: 2.65

Distance: 16.35

4−11−3: Behaviors and Animation

Behaviors are controls that affect the 3D display. In Electric, there are 3 types of behaviors available.

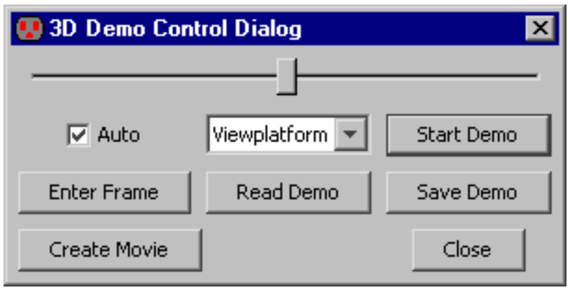
1. **Orbit Behavior** combines three basic mouse behaviors: zoom, pan and rotate. The left button rotates, the right button pans, and the middle button zooms. Click and drag to alter the display.
2. **3D Axis Behavior** is available when the 3D axis is shown. Clicking on the axis affects rotation (but not panning or zooming). This axis is not part of the standard Electric distribution and must be installed separately (see Section 1−5).
3. **Navigator Behavior** is controlled by special keys. Use the up/down/left/right arrow keys as shown in the table.

Key Press	Effect
DOWN Arrow	Move along −Z axis
UP Arrow	Move along +Z axis
CTRL + DOWN Arrow	Move along −Y axis
CTRL + UP Arrow	Move along +Y axis
ALT + LEFT Arrow	Move along −X axis
ALT + RIGHT Arrow	Move along +X axis
RIGHT Arrow	Rotate along −Y axis
LEFT Arrow	Rotate along +Y axis
CTRL + RIGHT Arrow	Rotate along −Z axis
CTRL + LEFT Arrow	Rotate along +Z axis
ALT + DOWN Arrow	Rotate along −X axis
ALT + UP Arrow	Rotate along +X axis

Animation

A 3D display can be animated by creating "key frames" along a time line. Interpolators examine the key frames and smoothly animate the 3D view. There are two types of interpolators: *simple* and *path*. Simple interpolators have a start and end frame, varying the view between them linearly. Path interpolators allow multiple key frames to combine into a single smooth animation.

Spline interpolators can be created and controlled with the **Capture Frame/Animate** command (in menu **Window / 3D Window**). To animate, you must create a sequence of key frames that define the view changes. Each key frame represents a different 3D view of the scene.



To control the animation, make changes to the display and click "Enter Frame". You can enter as many frames as you want and animate them later. The animated sequence is a "demo" that can be saved to disk and restored later for playback. Use the **Animate Sample Cell** command (in menu **Help / Samples**) to see an example of animation.

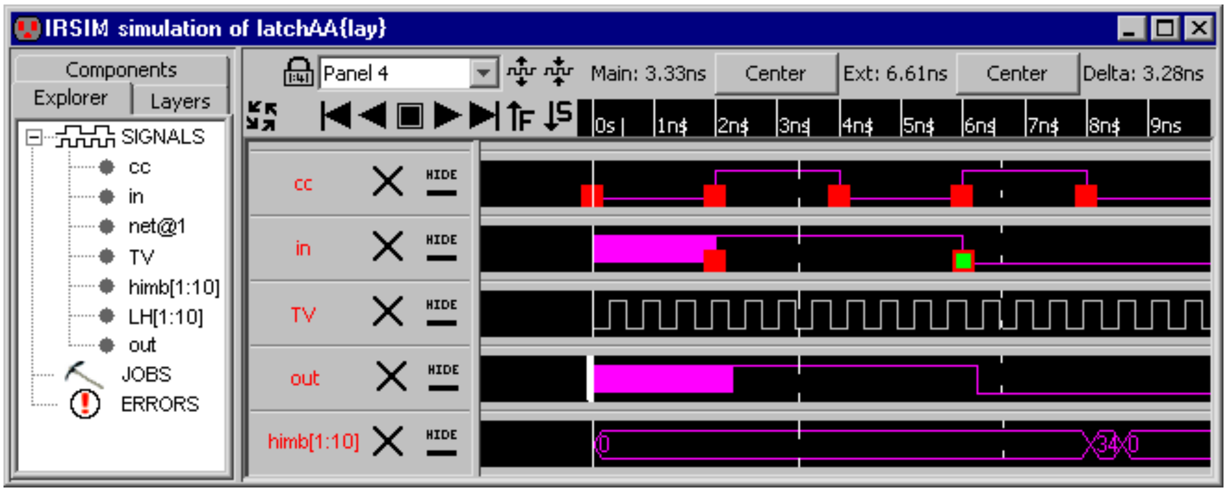
A QuickTime movie can be created by using the "Create Movie" button. For this option, the JMF plugin must be available (see Section 1−5).

Chapter 4: Display

4−12−1: Digital Waveform Windows

The waveform window is able to display digital simulation output. This simulation output can come from external simulators (such as Verilog and ArchSim) or built−in simulators (such as ALS and IRSIM). When displaying the results of external simulators, the system reads the simulation output and shows it. When internal simulators are displayed, you have the additional capability of changing the stimuli.

The digital waveform window looks like the picture below. Note also that there is a side bar with a cell explorer in the window, just like in all windows, but the explorer has a "SIGNALS" section that lists the signals found in the simulation.



Wave Panels

The waveform window contains a set of *panels*, each with a signal name and signal waveform. In each panel, signal names are shown on the left, and their waveform on the right. Between the name and the waveform are two control buttons:

- **Close** (an "X") to remove that panel from the waveform window.
- **Hide** to stop displaying the panel, but keep it available (it can be restored by selecting its name from the popup at the top of the waveform window).

The waveforms can be single signals or busses. Busses are collections of single signals that display integer values (for example, "himb[1:10]"). To expand a bus, and show its individual signals in separate panels, double−click on its name. To contract the bus (removing its individual signals), double−click on the bus name again.

Although the color of the waveforms is usually the same, it can vary with the strength of the signal. To enable such a display, check "Multistate display" in the "Simulators" preferences (in menu **File / Preferences...**, "Tools" section, "Simulators" tab). To control the actual colors used in multistate display, use the "Colors" preferences (in menu **File / Preferences...**, "Display" section, "Colors" tab) and set the colors for

"WAVEFORM: OFF STRENGTH", "WAVEFORM: NODE (WEAK) STRENGTH", "WAVEFORM: GATE STRENGTH", and "WAVEFORM: POWER STRENGTH".

You can select a signal by selecting either its name or the actual waveform. A selected signal is highlighted, and the selected panel is marked with a bold white line (see the "out" signal). Note that when you click on a signal, the equivalent network in the associated schematic or layout window is also highlighted.

You can rearrange the order of the signals by dragging their names to a new location.

You can add a new panel to the waveform window by double−clicking on its name in the "SIGNALS" area (or by dragging that name to the waveform part on the right). If the layout or schematics cell that produced the simulation is being displayed in another window, and the currently selected network in that window is found in the simulation output, then that output can be added to the waveform window with the **Add to Waveform in New Panel** command (in menu **Edit / Selection**).

The order of signals in the waveform window is saved in the original cell so that subsequent simulations will show the same signals.

Time Control

Two vertical cursors appear in the window, called "main" and "extension" (the extension cursor is dotted). Their time values and their difference are shown at the top of the window. You can click over the cursors and drag them to different time locations. You can also use the "Center" buttons to bring these cursors to the center of the display.

The time axis of the simulation window can be controlled with the appropriate **Window** menu commands. Use **Zoom Out** and **Zoom In** to scale the time axis by a factor of two. Use **Focus on Highlighted** to display the range between the main and extension cursors.

Besides controlling time with menu commands, you can also use the Pan and Zoom tools of the toolbar to change the view.

The pan tool lets you smoothly shift time when you click and drag. In the zoom tool, you zoom into an area by clicking and dragging out that area. To zoom out, hold the shift key and click in the center of the desired area.

The different panels in the waveform window are locked in time: they all show the same range of time, as shown at the top of the waveform window. If you click on the "time lock" button at the top of the waveform window (looks like a lock with the time on it: 🔒 🔓) then time is unlocked, and each panel has its own time scale. Now individual panels can show a different range of time than the rest.

Electric does crossprobing between the waveform window and an edit window with the original circuit. If the original circuit is being displayed, selection in the waveform window is mirrored in that cell. Also, whenever the main time cursor changes, the electrical state of the circuit is shown in that cell. Wires are colored differently according to their high/low/X/Z value in the simulation at that time. If you connect *Simulation Probe* nodes to any part of the circuit, those nodes light up with the appropriate color instead, which allows better visualization of activity patterns (see Section 7−6−3). You can control the colors used in crossprobing by using the "Colors" preferences (in menu **File / Preferences...**, "Display" section, "Colors" tab) and setting the colors for "WAVEFORM: CROSSPROBE LOW", "WAVEFORM: CROSSPROBE HIGH", "WAVEFORM: CROSSPROBE UNDEFINED", and "WAVEFORM: CROSSPROBE FLOATING".

For best visualization of the simulation activity, there is a set of VCR buttons to control an animation of the main time cursor. The play rate can be controlled by the "F" and "S" buttons which make it go faster or slower. As the time cursor sweeps across the waveform window, the original circuit can be seen to change levels.

These window functions apply to the digital simulation windows:

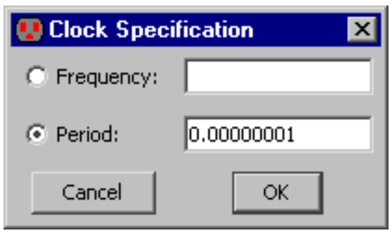
- **Window / Fill Window** make all data fit in window.
- **Window / Zoom Out** show twice as much time.
- **Window / Zoom In** show half as much time.
- **Window / Special Zoom / Focus on Highlighted** show from main to extension cursors.
- **Window / Pan Left** show earlier time.
- **Window / Pan Right** show later time.
- **"Pan" tool in tool bar** freehand drag of time.
- **"Zoom" tool in tool bar** drag area to zoom in, hold shift to zoom out.
- **"Measure" tool in tool bar** for measuring time.

Stimuli (for Built−in Simulators only)

When the waveform window displays the output of built−in simulators, you can set stimuli on the signals to affect the simulation. Each stimulus that you set is marked with a large red box at the time of the stimulus (see signals "cc" and "in"). You can select the stimuli by clicking on the red box. A selected stimulus has a green box in it (see the rightmost stimulus on signal "in").

To set stimuli, select either a waveform or the equivalent network in the original schematic or layout. Once selected, use the **Set Signal High at Main Time** (in menu **Tool / Simulation (Built−in)**) to make that signal go to "high" at the time indicated by the Main cursor. Use **Set Signal Low at Main Time** to set the selected signal "low", and use **Set Signal Undefined at Main Time** to set the selected signal "undefined" (X). Use the **Get Information about Selected Signals** command to show stimuli and other information on the selected signals.

Besides simple test vectors, the ALS simulator can also set clock patterns on the currently selected signal by using the **Set Clock on Selected Signal...** command. There are two ways to specify a clock: by frequency (in cycles per second) or period (in seconds).


Clock Specification

Frequency: [ ]

[✓ Period:] 0.00000001

Note that the clock cycles infinitely, but Electric generates simulation events to fill only the current waveform window. If you want more clock events generated, zoom−out the waveform window before issuing the clock command.

To remove the selected stimulus, use the **Clear Selected Stimuli** command. To remove all stimuli on a the selected waveforms, use **Clear All Stimuli on Selected Signals**. To remove all stimuli in the simulation, use **Clear All Stimuli**.

Once a set of stimuli has been established, you can save it to disk with the **Save Stimuli to Disk...** command. These stimuli can be restored later with the **Restore Stimuli from Disk...** command. Each built−in simulator has its own format for saving stimuli.

The "Simulators" preferences (in menu **File / Preferences...**, "Tools" section, "Simulators" tab), offers some controls for built−in simulators.

- "Auto advance time" requests that the main time cursor advance after each stimulus is added. This allows each stimulus added to occur at a new time.
- "Resimulate each change" requests that the simulator rerun the simulation after any change to the stimuli. Because the process of simulating a circuit can be costly, you might want to delay resimulation until all stimuli have been set. If you uncheck this item, you must issue the **Update Simulation Window** command to re−run the simulation.

Other Controls

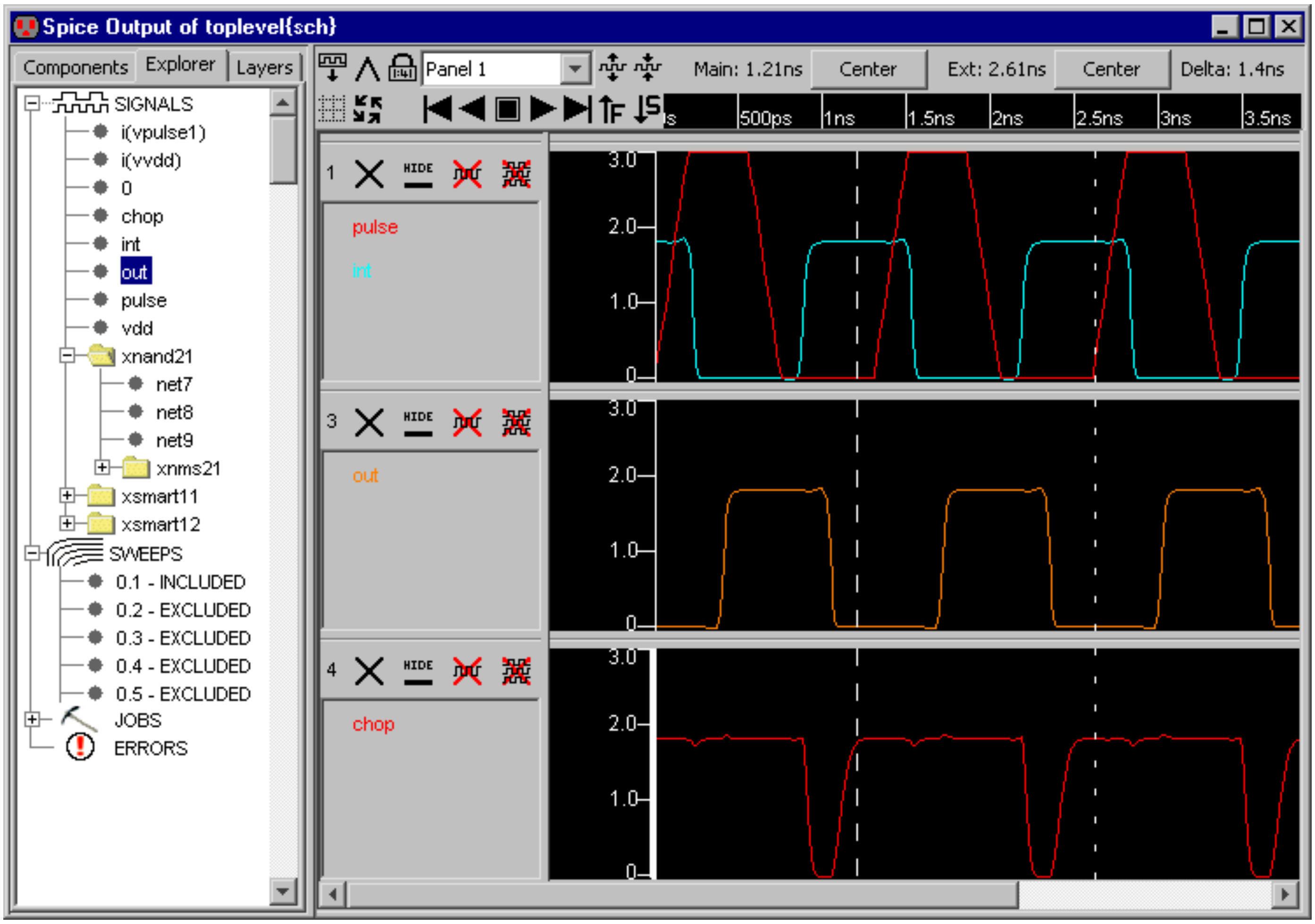
At the top of the waveform window, above the signal names, are many useful controls. Those relating to time have already been discussed. Here are the remaining buttons:

- **"Refresh"** Refreshing the simulation causes it to reload from the original source. In the case of external simulation, it is assumed that the simulation was re−run and the output file is different, so the simulation output file is re−read. In the case of built−in simulators, it is assumed that the original circuit has changed, so it is re−evaluated and reloaded into the simulator.
- **The Panel popup** This is a list of all panels, including the hidden ones. Selecting a panel from this list toggles its "hidden" state, making a visible one disappear, and making a hidden one reappear.
- **"Grow" and "Shrink"** These buttons, which show a waveform being stretched or squeezed, cause the minimum panel size to change. By shrinking the panel size, more of them can fit in the window without having to use a slider to access them.

4−12−2: Analog Waveform Windows

The waveform window is able to display analog simulation output. This simulation output comes from external simulators (such as Spice). When the system is asked to display the results of an external simulation, it reads the simulation output and shows it.

The analog waveform window looks like the picture below. Note that there is a side bar with a cell explorer in the window, just like in all windows, but the explorer has a "SIGNALS" section that lists the signals found in the simulation (and optionally a "SWEEPS" section if swept data was found).



Wave Panels

The waveform window contains a set of *panels*, each with one or more signals and waveforms. In a panel, signal names are shown on the left, and their waveform on the right. Above the signal names in each panel are 5 names and controls:

- **Panel number** each panel is numbered so that it can be hidden and retrieved.
- **Close** (an "X") to remove the panel from the waveform window.
- **Hide** to stop displaying the panel, but keep it available (it can be restored by selecting its name from the popup at the top of the waveform window).
- **Remove Signal** remove the selected signal from the panel (the DELETE key works for this, too).
- **Remove All Signals** remove all signals from the panel.

You can select a signal by selecting either its name or the actual waveform. Note that when you click on a signal, the equivalent network in the associated schematic or layout window is also highlighted.

You can rearrange the order of the signals by dragging their names to their desired location. You can change the color of a signal by right−clicking on its name.

You can add a signal to the list by double−clicking on its name in the "SIGNALS" area (or by dragging that name to the waveform part on the right). The signal will be added to the highlighted panel (the one with the bold vertical axis). You can create a new panel, with no signals in it, by clicking on the button in the upper−left of the waveform window (looks like: a panel−with−waveform icon being dropped down).

If the simulation had sweeps, those values are shown in the cell explorer in the "SWEEPS" area. You can right−click on a sweep and choose to include or exclude it from the display. You can also request that a sweep signal be highlighted. Right−clicking on the "SWEEPS" icon lets you include or exclude all of them.

If the layout or schematics cell that produced the simulation is being displayed in another window, and the currently selected network in that window is found in the simulation output, then that output can be added to the waveform window with the **Add to Waveform in New Panel** command (in menu **Edit / Selection**). The command **Add to Waveform in Current Panel** overlays the signal on top of others in the currently selected waveform panel.

The order of signals in the waveform window is saved in the original cell so that subsequent simulations will show the same signals.

Time Control

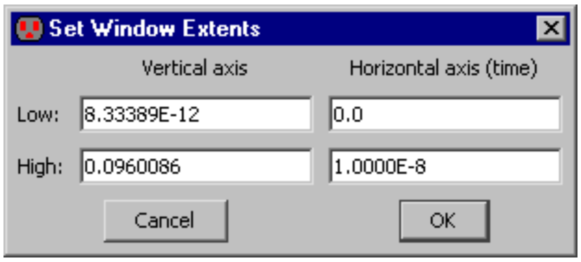
Two vertical cursors appear in the window, called "main" and "extension" (the extension cursor is dotted). Their time values and their difference are shown at the top of the window. You can click over the cursors and drag them to different time locations. You can also use the "Center" buttons to bring these cursors to the center of the display.

The time axis of the simulation window can be controlled with the appropriate **Window** menu commands. Use **Zoom Out** and **Zoom In** to scale the time axis by a factor of two. Use **Focus on Highlighted** to display the range between the main and extension cursors.

Besides controlling time with menu commands, you can also use the Pan and Zoom tools of the toolbar.

The pan tool lets you smoothly shift time when you click and drag. In the zoom tool, you zoom into an area by clicking and dragging out that area. To zoom out, hold the shift key and click in the center of the desired area.

You can control the horizontal and vertical range precisely by double−clicking in the vertical scale area. The dialog lets you type exact values into the ranges.



	Vertical axis	Horizontal axis (time)
Low:	8.33389E-12	0.0
High:	0.0960086	1.0000E-8

The different panels in the waveform window are locked in time: they all show the same range of time, as shown at the top of the waveform window. If you click on the "time lock" button at the top of the waveform window (looks like a lock with the time on it:) then time is unlocked, and each panel has its own time scale. Now individual panels can show a different range of time than the rest.

A set of VCR buttons is available to animate the main time cursor. The play rate can be controlled by the "F" and "S" buttons which make it go faster or slower. As the time cursor sweeps across the waveform window, the original circuit can be seen to change levels.

These window functions apply to the analog simulation windows:

- **Window / Fill Window** make all data fit in window.
- **Window / Zoom Out** show twice as much time.
- **Window / Zoom In** show half as much time.
- **Window / Special Zoom / Focus on Highlighted** show from main to extension cursors.
- **Window / Pan Left** show earlier time.
- **Window / Pan Right** show later time.
- **"Pan" tool in tool bar** freehand drag of time.
- **"Zoom" tool in tool bar** drag area to zoom in, hold shift to zoom out.
- **"Measure" tool in tool bar** for measuring time.

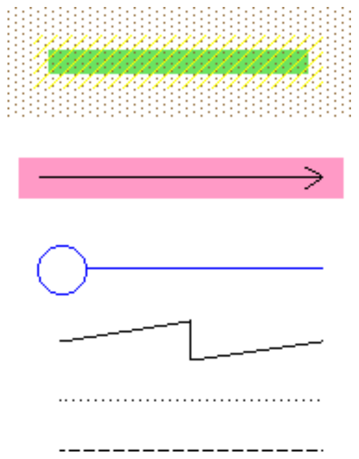
Other Controls

At the top of the waveform window, above the signal names, are many useful controls. Those relating to time have already been discussed. Here are the remaining buttons:

- **"Refresh"** Rereads the simultion output file and updates the display. If the simulation has been re−run, and the output file is different, then this button shows the new data.
- **"Show Vertices"** Displays dots on the vertices of the waveforms. The button toggles between showing and not−showing the vertex dots.
- **"Show Grid"** Displays a grid in the waveform panels. The button toggles between showing and not−showing the grid.
- **The Panel popup** This is a list of all panels, including the hidden ones. Selecting a panel from this list toggles its "hidden" state, making a visible one disappear, and making a hidden one reappear.
- **"Grow" and "Shrink"** These buttons, which show a waveform being stretched or squeezed, cause the minimum panel size to change. By shrinking the panel size, more of them can fit in the window without having to use a slider to access them.

Chapter 5: Arcs

5−1: Introduction to Arcs



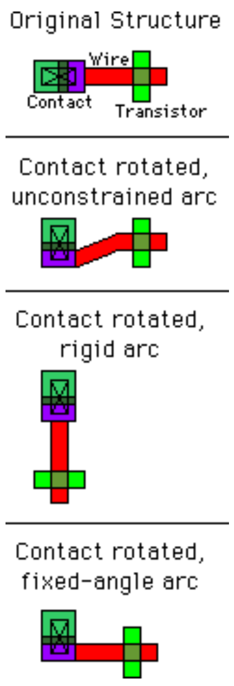
The arcs in a circuit are much more than simple connecting wires. They can take many different forms according to the needs of the design environment. In schematics, arcs can be negated, directional, zig−zag, and more. In layout, they can be directional and extended by half of their width.

The most important property of an arc is its ability to remain connected when physical changes are made to the circuit. Constraining properties provide for intelligent circuit layout.

Electric allows you to control how layout changes when the circuit is modified. This is done by placing *constraints* on the arcs that react to node changes. Electric has a set of four constraints that, although not complete, have been found to be useful in circuit design.

5−2−1: Rigid and Fixed−Angle Arcs

The first constraint in Electric is the *rigid* constraint. When an arc is made rigid, it cannot change length. If a node on either end is moved, the other node and the arc move by the same amount. Besides keeping a constant length, rigid arcs attach in a fixed way to their nodes. This means that if the node rotates or mirrors, the arc spins about so that the overall configuration does not change. Without this rigidity constraint, arcs simply stretch and rotate to keep their connectivity.

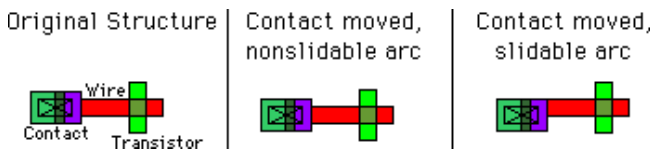

Original Structure
Wire
Contact
Transistor
Contact rotated, unconstrained arc
Contact rotated, rigid arc
Contact rotated, fixed−angle arc

The second constraint, which is used only if an arc is not rigid, is the *fixed−angle* constraint. This constraint forces a wire to remain at a constant angle (usually used to keep horizontal and vertical wires in their Manhattan orientations). For example, if a vertical fixed−angle arc connects two nodes, and the bottom node moves left, then the arc and the top node also move left by the same amount. If that bottom node moves down, the arc simply stretches without affecting the other node. If the bottom node moves down and to the left, the arc both moves and stretches. Rotation of nodes causes no change to fixed−angle arcs unless the arc is connected to an off−center port, in which case a slight translation and stretch may occur.

Most IC layout is done with Manhattan geometry. If you suspect that some of your wires have become skewed, use the **Show Nonmanhattan** command (in menu **Edit / Cleanup Cell**).

5−2−2: Slidable Arcs

Another constraint, available only for nonrigid arcs, is *slidability*. When an arc is slidable, it may move about within its port. To understand this fully, you should know exactly where the arc *endpoint* is located. Most arcs are defined to extend past the endpoint by one−half of their width. This means that the arc endpoint is centered in the end of the arc rectangle. If the arc is 2 wide, then the endpoint is indented 1 from the edge of its rectangle. All arc endpoints must be inside of the port to which they connect. If the port is a single point, then there is no question of where the arc may attach. If, however, the port has a larger area, as in the case of contacts, then the arc can actually connect in any number of locations.



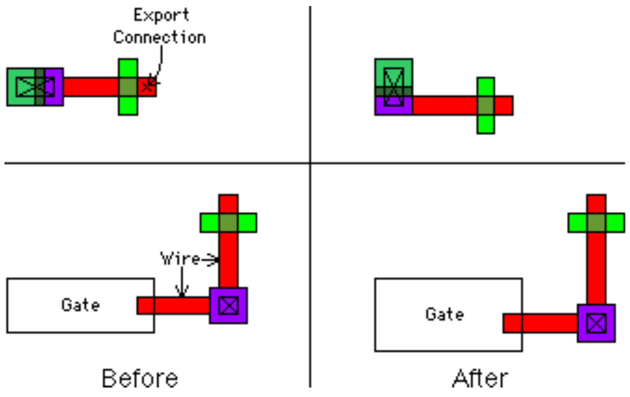
Slidable arcs may adjust themselves within the port area rather than move. For example, if a node's motion is such that the arc can slide without moving, then no change occurs to the arc or to the other node. Without the slidable constraint, the arc moves to stay connected at the same location within the port. Slidability propagation works both ways, because if an arc moves but can slide within the other node's port, then that node does not move. Note that slidability occurs only for complete motions and not for parts of a motion. If the node moves by 10 and can slide by 1, then it pushes the arc by the full 10 and no sliding occurs. In this case, only motions of 1 or less will slide.

Because ports have area, and because arcs end somewhere inside of that area, the actual ending point can vary considerably. If the arc is at the far side of the port, it may protrude out of the far side of the node, causing unwanted extra geometry. You can shorten an arc so that its endpoint is at the closest side of the port with the **Shorten Selected Arcs** command (in menu **Edit / Cleanup Cell**).

5−2−3: Constraint Propagation

The last of Electric's constraints is the only one that is not actually programmable by the user.

This is the constraint that all arcs must stay in their ports, even across hierarchical levels of design. When a node in a cell moves, and has an export on it, all the ports on instances of that cell also change. The constraint system therefore adjusts all arcs connected to those instances, and follows their constraints. If those constraints change nodes with exports in the higher−level cell, then the changes propagate up another level of hierarchy.



This bottom−up propagation of changes guarantees a correctly connected hierarchy, and allows top−down design. Users can create skeleton cells that are mostly empty and contain only exports on unconnected nodes. They can then do high−level design with these skeleton cell instances. Later, when circuitry is placed in the cells, or when layout views are substituted for the skeletons, the constraint system will maintain proper connectivity in all higher levels of hierarchy.

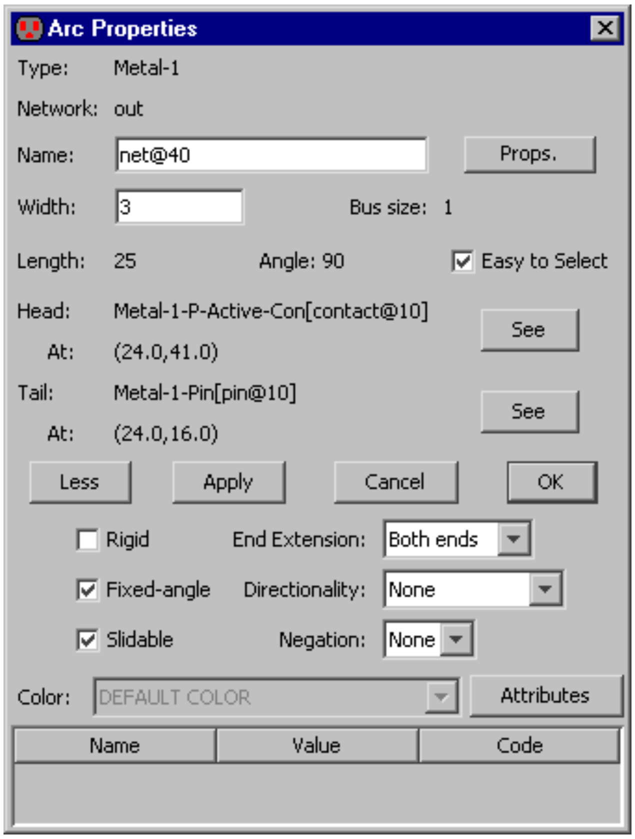
The hierarchical−propagation aspect of the constraint system leaves open the possibility of an overconstrained situation. For example, if two different cell instances are connected to each other with two rigid wires, and one connection point moves, then it is not possible to keep both wires rigid. Electric jogs an arc, converting it into three arcs that zig−zag, to retain the connection. Although connectivity is retained, the geometry may be in the wrong place, causing unexpected changes to the circuit. Users are encouraged to examine the hierarchy to make sure that arbitrary hierarchical changes do not cause undetected damage to the layout. Electric will warn you of any changes which affect undisplayed cells farther up the hierarchy.

5−3: Setting Constraints

The two most common constraints, rigid and fixed−angle (see Section 5−2−1), can be controlled from the **Edit / Arc** menu. When the **Rigid**, **Non Rigid**, **Fixed Angle**, and **Not Fixed Angle** commands are issued, all of the currently highlighted arcs have those constraints set.

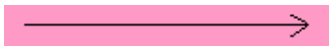
In order to set slidability (see Section 5−2−2), select a single arc and issue the **Object Properties...** command (in menu **Edit / Properties**).

At the bottom of the arc properties dialog, when the "More" button has been pressed, are check boxes that control constraints. This is the only way to affect the slidable constraint (which is not very commonly used).


Arc Properties

Type: Metal-1
Network: out
Name: net@40
Width: 3
Bus size: 1
Length: 25
Angle: 90
☑ Easy to Select
Head: Metal-1-P-Active-Con[contact@10]
At: (24.0,41.0)
Tail: Metal-1-Pin[pin@10]
At: (24.0,16.0)
☐ Rigid
End Extension: Both ends
☑ Fixed-angle
Directionality: None
☑ Slidable
Negation: None
Color: DEFAULT COLOR

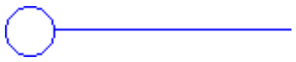
5–4–1: Directionality



For documentation purposes, it is possible to display a *directional* arrow on arcs to indicates flow. This property can be changed with the **Toggle Directionality** command (in menu **Edit / Arc**). It may also be controlled by the **Object Properties...** dialog (in menu **Edit / Properties**).

The controls in the **Object Properties...** dialog offer the option of placing the arrow head on either end, both ends, or neither end. This allows arbitrary combinations of arrow heads and bodies to display arbitrarily intricate directionality schemes.

5–4–2: Negation

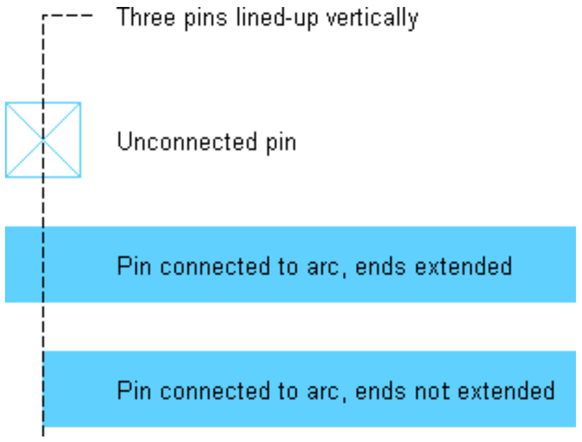


Arcs in the Schematic technology may be *negated*, which causes them to have a bubble drawn where they attach to schematic elements. This property can be changed with the **Toggle Port Negation** command (in menu **Edit / Technology Specific**). It may also be controlled by the **Object Properties...** dialog (in menu **Edit / Properties**). Note that you can select an arc to toggle negation (which leaves the system to guess which end you want to negate) or you can select a node and port (in which case, the arc attached to that port is negated).

Note that the **Object Properties...** dialog offers precise control of the negating bubbles, allowing you to specify which ends have the bubbles on them. Negated arcs make no sense in layout technologies and are ignored.

5−4−3: End Extension

All arcs are drawn so that their geometry extends beyond their endpoints by one−half of their width. This property can be set or reset with the **Toggle End Extension of Head** and **Toggle End Extension of Tail** commands (in menu **Edit / Arc**). It may also be controlled by the **Object Properties...** dialog (in menu **Edit / Properties**).


Three pins lined-up vertically
Unconnected pin
Pin connected to arc, ends extended
Pin connected to arc, ends not extended

5−4−4: Naming

Another property of an arc is its name. This is a character string that is displayed on the arc and used to name the electrical network connected to that arc. The "Name" field in the **Object Properties...** dialog allows you to specify this property, which is then displayed on the arc. Note that creating exports is another way of naming a network. See Section 6−9−2 for more on network naming.

All arcs are named in Electric, so if you don't give it a name, one will be assigned. These names, which typically take the form "object@number" are *temporary* names, and are distinguished from the names given by the user.

Arc names can be quite complex when applied to busses. The names can be indexed, aggregated, and otherwise be used to describe multiple signals. See Section 6−9−3 for more on bus naming.

5−4−5: Curvature

An unusual arc property, used only in circular geometry, is curvature. Although most arcs cannot handle curvature, those in the Artwork and Round CMOS ("rcmos") technologies can.

The **Curve through Cursor** command (in menu **Edit / Arc**) requests that the currently highlighted arc curve in such a way that it passes through the location of the cursor. The **Curve about Cursor** command requests that the currently highlighted arc curve between its endpoints such that the center of curvature is at the location of the cursor. After issuing these commands, click and drag to see how the arc will curve.

The **Remove Curvature** command makes the arc straight.

5−5: Default Arc Properties

The "New Arcs" preferences (in menu **File / Preferences...**, "General" section, "New Arcs" tab) lets you control the arc creation process. It does not affect existing arcs: they have already been created.

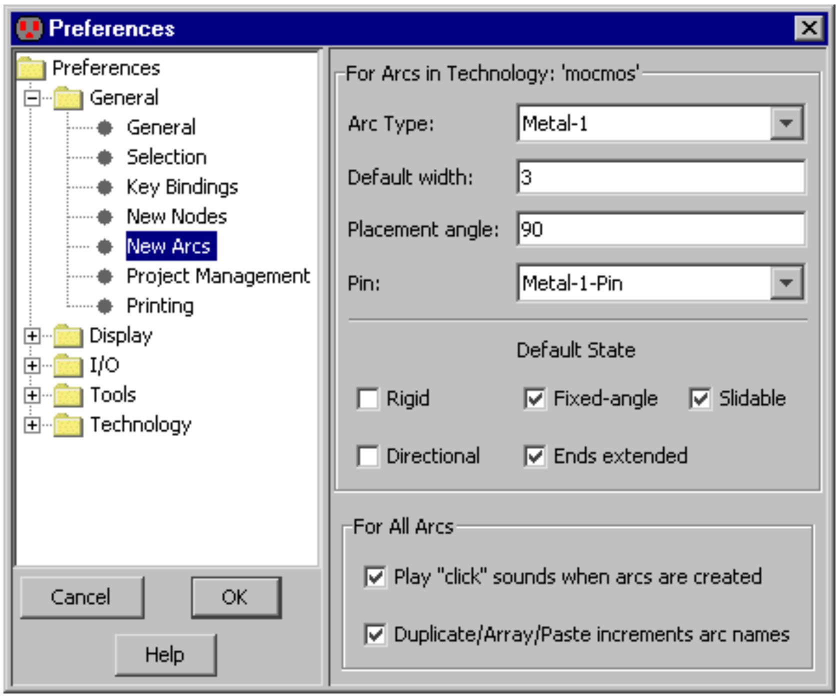
The top part of the dialog allows you to set defaults for specific types of arcs in the current technology. You select the "Arc Type" and then set defaults for it (such as the "Default width").

The "Placement angle" is the granularity for running this type of arc (in degrees). A value of 90 lets arcs run at 0, 90, 180, or 270 degrees: manhattan geometry. A value of 45 lets it run at any of 8 angles (useful for schematics). A value of 0 lets it run at any angle (used in artwork).

The "Pin" is the node that gets used for connecting two of these arcs. It is typically a "Pin" node (see Section 7−1−1). If changed to a node with geometry (such as a contact node) then these contacts will be placed at the bends of this arc.

The checkboxes in the "Default State" section have these meanings:

- Rigid − whether the arc is rigid in length and relationship to its nodes (see Section 5−2−1).
- Fixed−angle − whether the arc stays at the same angle when one end moves (see Section 5−2−1).
- Slidable − whether the arc slides around in its port (see Section 5−2−2).
- Directional − whether the arc has an arrow drawn on it (see Section 5−4−1).
- Ends extended − whether the arc extends past its endpoint by half its width (see Section 5−4−3).



For Arcs in Technology: 'mocmos'

Arc Type: Metal-1

Default width: 3

Placement angle: 90

Pin: Metal-1-Pin

Default State

☐ Rigid ☑ Fixed-angle ☑ Slidable

☐ Directional ☑ Ends extended

For All Arcs

☑ Play "click" sounds when arcs are created

☑ Duplicate/Array/Paste increments arc names

The bottom portion of the dialog has controls for all arcs.

- "Play 'click' sounds when arcs are created" − plays a sound to confirm arc creation. The sound is a single click for one arc, a double−click for two arcs, and a triple−click for three or more arcs.
- "Duplicate/Array/Paste increments arc names" − sets whether the name on an arc should be kept unique by auto−incrementing after this arc has been duplicated, arrayed, or pasted.

Chapter 6: Advanced Editing

6−1: Making Copies

Once you have created a collection of objects, it may be desirable to have other identical copies. There are two ways to do this: by duplication, and by cut−and−paste.

Duplication

The **Duplicate** command (in menu **Edit**) makes a copy of the selected nodes and arcs. After issuing this command, you can move the cursor to any location and click to place the copy. While moving the cursor, an outline of the duplicated objects is shown (as well as the amount of motion).

If you have disabled "Move after Duplicate" (in the "New Nodes" preferences, in menu **File / Preferences...**, "General" section, "New Nodes" tab) then the duplicated objects are placed immediately without dragging.

If any of the nodes have exports on them, they are not duplicated (unless "Duplicate/Array/Paste copies exports" is set in "New Nodes" tab).

The **Duplicate** command forces newly created nodes and arcs to have unique names. This means that if any nodes or arcs are named (using the **Object Properties...** command, in menu **Edit / Properties**) and then duplicated, the new ones will have different names (specifically, the old names with numbers appended or modified).

Cut−and−Paste

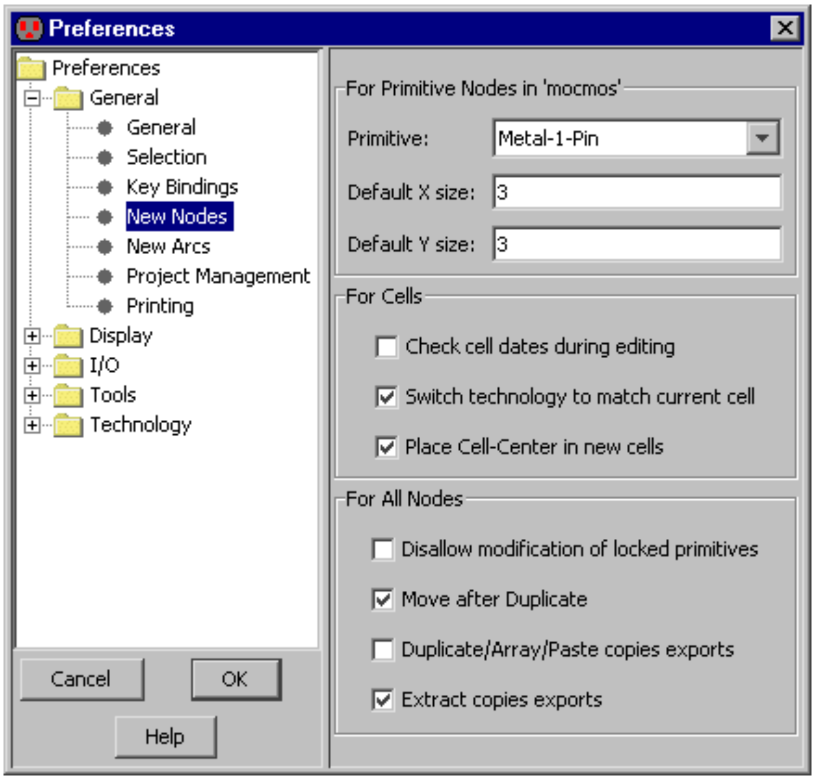
Another way to make copies of nodes and arcs is with the cut−and−paste commands. The **Copy** and **Cut** commands (in menu **Edit**) copy the currently selected nodes and arcs to a special buffer. **Cut** also removes the objects after copying them. The **Paste** command then copies the objects from the special buffer to the display. After issuing this command, an outline of the pasted objects attaches to the cursor. When you click, the objects are placed at that location. You can right−click during the paste drag to affect the location, and to abort the paste.

Note that if you copy a node or arc and then select another before pasting, then the copied object will replace the selected object (changing its type and other properties, similar to the **Change...** command, in menu **Edit**, see Section 6−6). If you want the **Paste** command to make a second copy, be sure that nothing is selected when you issue the command. Thus, duplicating an object cannot be done by issuing a **Copy** and then a **Paste**. You must do a **Copy**, then deselect the object, then do a **Paste**.

6−2: Creation Defaults

The **Duplicate** command is useful because a node may have been modified (rotated, scaled, etc.) and duplication preserves all of those changes. Using **Copy** and **Paste** does the same thing. Another way to create nodes that are nonstandard is to set creation defaults.

To do this, use the "New Nodes" preferences (in menu **File / Preferences...**, "General" section, "New Nodes" tab). The top part of the dialog controls primitive nodes. You can change the default size of any primitive node in the current technology by choosing the node and changing the values.



For Primitive Nodes in 'mocmos'

Primitive: Metal-1-Pin

Default X size: 3

Default Y size: 3

For Cells

☐ Check cell dates during editing

☑ Switch technology to match current cell

☑ Place Cell-Center in new cells

For All Nodes

☐ Disallow modification of locked primitives

☑ Move after Duplicate

☐ Duplicate/Array/Paste copies exports

☑ Extract copies exports

The middle section of the dialog controls cells. The check box "Check cell dates during editing" requests that date information be used to ensure a proper circuit building sequence. When this box is checked, warning messages will be issued when editing a cell that has more recent subcell instances. Electric tracks cell creation and revision dates, and this information can be displayed with the **Describe this Cell** command and others in menu **Cell / Cell Info** (see Section 3−7−1).

The check box "Switch technology to match current cell" requests that the current technology automatically change whenever the current cell changes, so that the two match.

The check box "Place Cell−Center in new cells" requests that all newly created cells have a Cell−Center node placed at the origin (see Section 3−3 for more on Cell centers).

The bottom part of the dialog applies to all nodes.

The check box "Disallow modification of locked primitives" requests that all lockable primitive node instances be anchored. Once locked, these nodes cannot be created, deleted, or modified in any way. Typically, only primitives in "array" technologies are lockable (such as the FPGA technology, see Section 7−6−2), presuming that these components will be used to define the fixed circuitry that is then customized. Design of the fixed circuitry is done with this lock off, and then the customization phase is done with this lock on.

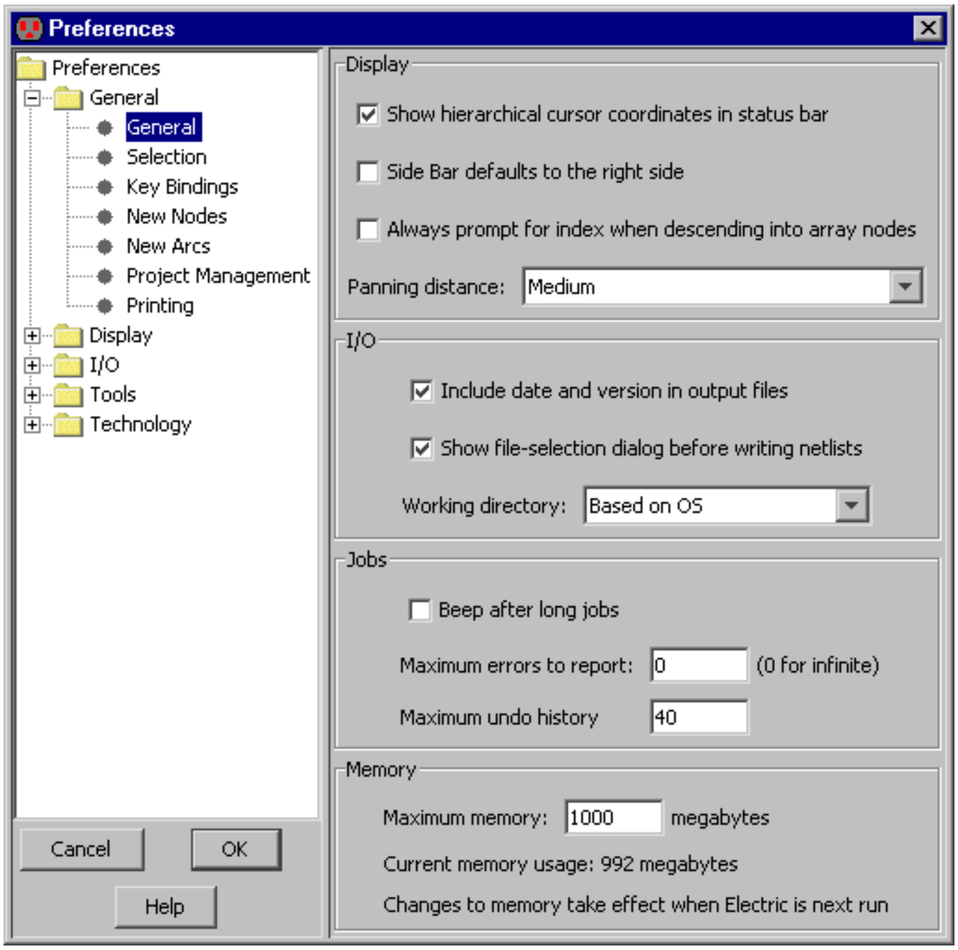
The check box "Move after Duplicate" allows duplicated objects to be positioned interactively. This is the default condition. However, if this is unchecked, then the **Duplicate** command (in menu **Edit**) will place a copy automatically, without allowing the new location to be specified by the cursor.

The check box "Duplicate/Array/Paste copies exports" requests that these node−copying operations also copy their exports. This includes the **Duplicate**, **Array**, and **Paste** commands (in menu **Edit**) . See Section 6−4 for more on arrays.

The check box "Extract copies exports" requests that extraction of cell instances also copy the exports. Extraction is done with the **Extract Cell Instance** command (in menu **Cell**). See Section 3−8 for more on extraction.

6−3: Preferences

The **Preferences...** command (in menu **File**) contains dozens of panels for controling preferences. You can also see the preferences dialog by using the preferences icon from the tool bar.



Display

[✓] Show hierarchical cursor coordinates in status bar

[ ] Side Bar defaults to the right side

[ ] Always prompt for index when descending into array nodes

Panning distance: Medium

I/O

[✓] Include date and version in output files

[✓] Show file-selection dialog before writing netlists

Working directory: Based on OS

Jobs

[ ] Beep after long jobs

Maximum errors to report: 0 (0 for infinite)

Maximum undo history 40

Memory

Maximum memory: 1000 megabytes

Current memory usage: 992 megabytes

Changes to memory take effect when Electric is next run

The left side of the Preferences dialog is a tree−structured list of all of the different preference panels. The right side is the actual preference panel, which changes according to the panel requested. Below the list of panels is a "Help" button which takes you to the proper manual page which explains that panel.

Preferences are stored permanently on your computer and are remembered each time you run Electric. The actual location of this information varies with each operating system.

- **Windows:** In the registry. Look in: HKEY_CURRENT_USER / Software / JavaSoft / Prefs / com / sun / electric.

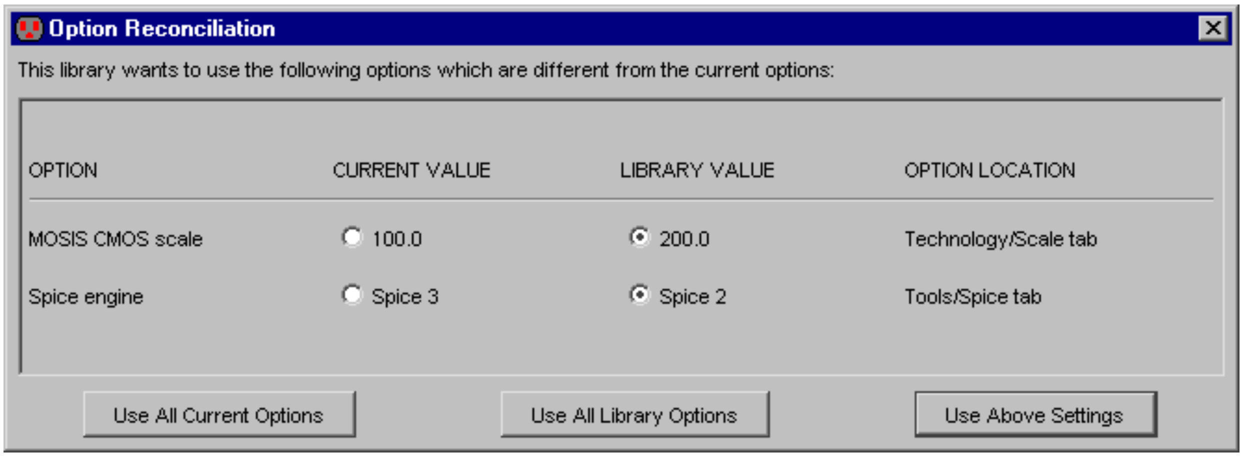
- **UNIX/Linux:** In your home directory. Look in: ~/.java/.userPrefs/com/sun/electric
- **Macintosh:** In your home directory, under Library/Preferences. Look at: ~/Library/Preferences/com.sun.electric.plist

You can delete the appropriate data to reset Electric to its "factory" state.

You can also export your preferences with the **Preferences...** command (in menu **File / Export**) which will write an XML file with preference information. This XML file can be read back into Electric's preferences with the **Preferences...** command (in menu **File / Import**).

There are two types of preferences: *appearance* and *meaning*. Appearance preferences affect only the way Electric appears when run, and do not affect the actual circuitry. Meaning preferences affect the interpretation of the data, and therefore their settings are critical to your design.

Electric stores meaning preferences in every library file so that, when read back in, the preferences can be reconciled with current settings. When a library is read that has different meaning preferences, this dialog appears:



This library wants to use the following options which are different from the current options:

OPTION	CURRENT VALUE	LIBRARY VALUE	OPTION LOCATION
MOSIS CMOS scale	100.0	[✓ 200.0]	Technology/Scale tab
Spice engine	Spice 3	[✓ Spice 2]	Tools/Spice tab

You must choose whether you want to use the preference value from the library (recommended) or the current setting. This can be done on an individual−preference basis, or for all preferences that conflict.

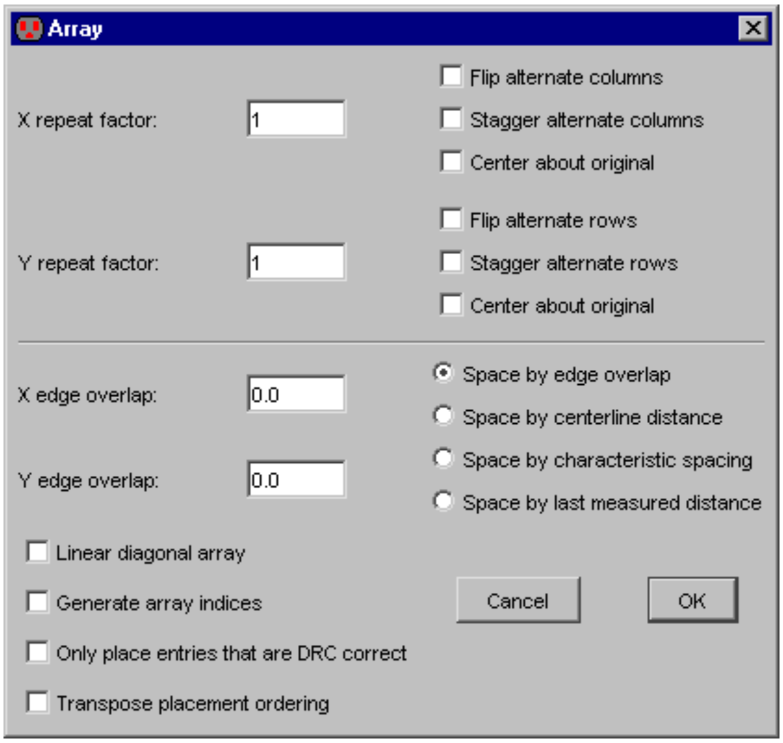
6−4: Making Arrays

If one copy is not enough, you may want an array of objects.

The **Array...** command (in menu **Edit**) takes the currently highlighted objects and replicates them many times. You specify the number of replications in the X and Y directions and the geometry is arrayed.

Arrays are generated by X (row) with Y (column), following a raster scan order. If you request that alternate rows or columns be flipped, then they are mirrored in the direction of repetition. If you request that alternate rows or columns be staggered, then each element is offset by an alternating amount. If you request that the rows or columns be centered, then the original circuitry will be placed in the middle of the array instead of the corner. If the X or Y values are negative, then the array is laid out backwards (replications are placed in the reverse direction).

There are four ways to specify spacing: *edge overlap*, *centerline distance*, *characteristic spacing*, or *measured distance*. The edge overlap amounts indicate the amount by which the rows and columns will be squeezed together (zero overlap causes the each arrayed copy to touch the next one, negative overlap can be specified to spread the objects apart). Centerline distance is the distance between object centers, and defaults to the size of the selected objects (which causes the copies to touch). Characteristic spacing is an amount that is set for specific cells (see Section 3−7−3). If a cell with a characteristic spacing is arrayed, that value can be used. Finally, the last measured distance can be used to determine the array spacing (for more on measuring, see Section 4−7−4).


Array

X repeat factor: 1
Y repeat factor: 1

☐ Flip alternate columns
☐ Stagger alternate columns
☐ Center about original
☐ Flip alternate rows
☐ Stagger alternate rows
☐ Center about original

X edge overlap: 0.0
Y edge overlap: 0.0

[✓ Space by edge overlap]
Space by centerline distance
Space by characteristic spacing
Space by last measured distance

☐ Linear diagonal array
☐ Generate array indices
☐ Only place entries that are DRC correct
☐ Transpose placement ordering

The "Linear diagonal array" check box indicates that the array is linear (one of the repeat factors must be 1) but that both spacing rules will be applied. This therefore creates a single line that runs diagonally.

The "Generate array indices" check box requests that the array entries be drawn with index information. When this is checked, array entries are labeled with the index of each entry. The original copy is labeled "0−0" and the copy to its right is labeled "1−0". These names are simply visual tags that have no bearing on the contents (use the **Object Properties...** command, in menu **Edit / Properties**, to set or remove these names).

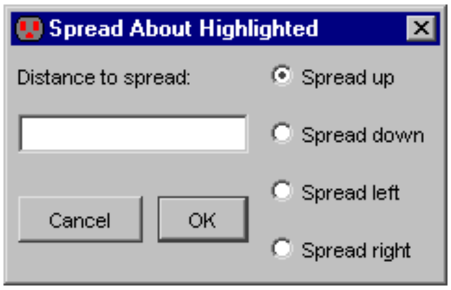
The "Only place entries that are DRC correct" check box requests that array entries only be placed where they do not create design−rule violations. This option is only available if a single node is being arrayed. After the array is created, the design−rule checker is run on each entry, and if it causes an error, it is removed.

The "Transpose placement ordering" check box requests that array placement go by column instead of by row. This is useful if the arraying includes names which are being auto−incremented in the array. By transposing the order of arraying, the names will run in the orthogonal direction.

6–5: Spreading Circuitry

When a large amount of circuitry has been placed too close together or too far apart, Electric's constraint system can help. All that is necessary is to make all arcs in an area rigid and then move one node. Of course, you may have to move more than one node if the one you pick is not connected to everything else you want to move. Also, you must make sure that arcs connecting across the area boundary are nonrigid. Finally, setting arc rigidity should be done temporarily so that it does not spoil an existing constraint setup. All these operations are handled for you by the **Spread...** command (in menu **Edit / Move**).

With the **Spread...** command, the highlighted node is a focal point about which objects move. A dialog is presented in which an amount and a direction (up, down, left, or right) are specified. An infinite line is passed through the highlighted node's center and everything above, below, to the left of, or to the right of the line is moved by the specified amount.

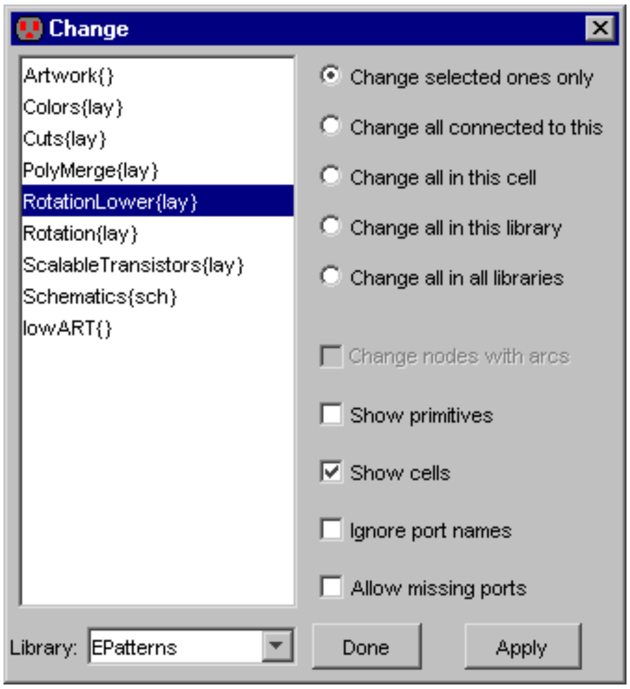


[✓ Spread up] Spread down Spread left Spread right

Negative spread distances compact the circuit.

6−6: Replacing Circuitry

The **Change...** command (in menu **Edit**) removes the currently highlighted node or arc and replaces it with a new one of a different type. This same effect can be had by copying one object and then pasting it onto another (see Section 6−1). A dialog is presented in which the possible replacements are shown. For node changing, you can choose to show primitives from the current technology, cells from the current library, or both.



Artwork{}
Colors{lay}
Cuts{lay}
PolyMerge{lay}
RotationLower{lay}
Rotation{lay}
ScalableTransistors{lay}
Schematics{sch}
lowART{}

[✓ Change selected ones only]
Change all connected to this
Change all in this cell
Change all in this library
Change all in all libraries

☐ Change nodes with arcs
☐ Show primitives
☑ Show cells
☐ Ignore port names
☐ Allow missing ports

Library: EPatterns

When replacing an arc, the existing nodes on either end must be able to reconnect to the new type of arc. If "Change nodes with arcs" is checked, nodes will be changed to allow the new type of arc to remain connected.

When replacing a node, the existing arcs on it must be able to reconnect properly to the new node. However, the sizes of the replaced object can be different, and the layout will be adjusted. Electric determines which ports on the replaced node to use by examining the port names and locations. If the ports are aligned correctly but not named the same, this matching will fail. Check "Ignore port names" to disable name matching and use only position information. If the new node is missing essential ports, such that existing wires cannot be reconnected, then the change will fail (unless "Allow missing ports" is checked).

Besides replacing the currently highlighted node or arc ("Change selected ones only"), it is also possible to specify replacement of many other objects.

- "Change all connected to this" requests that other objects of the same type which are connected to the highlighted ones will be changed.
- "Change all in this cell" requests that all other objects of the same type in this cell will be changed.
- "Change all in this library" requests that all other objects of the same type in the current library will be changed.
- "Change all in all libraries" requests that all other objects of the same type in every library will be changed.

This is a modeless dialog: it can remain up while other editing is being done. Click "Done" to dismiss it, and "Apply" to make a change.

Note that some Schematic nodes use parameters to further describe them. For example, an electrolytic capacitor is really just a capacitor with the "electrolytic" parameter on it. Therefore, you can change a node into a capacitor, but not an electrolytic capacitor, because it is not in the list. To change a capacitor into an electrolytic capacitor, paste an electrolytic capacitor onto it. Besides capacitors, parameters can be found on diodes, transistors, sources, and two−ports (the four−connection primitives such as VCCS).

Chapter 6: Advanced Editing

6−7: Undo Control

Electric has an undo mechanism that tracks all changes made during a session. When a command is issued, it and its side effects are stored.

The **Undo** command (in menu **Edit**) reverses the last change made (this includes any changes that may have been made by other tools). Multiple uses of the **Undo** command continue to undo further back. The **Redo** command redoes changes, up to the most recent change made.

You can also use the undo (counterclockwise) and redo (clockwise) icons from the tool bar.

Electric stores only the last 40 changes, so anything older than that cannot be undone. To increase the number of changes that are saved, use the "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab), and change the "Maximum undo history" field. To see a history of changes that were made, use the **Show Undo List** command (in menu **Edit / Info**).

In Electric, almost every command is undoable, but there are some exceptions. Commands that write disk files are not undoable, because Electric would not be so presumptuous as to delete a disk file. Also commands that make vast changes (such as library deletion) are not undoable.

Another useful command in for controlling changes being made is **Repeat Last Action** (in menu **Edit**). This repeats the last command, but only works for commands that can sensibly be repeated.

6−8−1: Understanding Text

There are a number of ways to place text in a circuit.

- Each unexpanded instance of a cell has text that describes it, and its ports.
- Each export has a text label.
- Nodes and arcs can be named (with **Object Properties...**) so that they have text on them. They can also have additional attributes that appear as text.
- Certain primitive nodes (such as the Flip−Flop component of the Schematic technology) have text as an integral part of their image.
- It is even possible to create a special node that is only text (with some of the commands under the "Misc" entry of the component menu: "Annotation Text", "Spice Code", "Verilog Code", and "Verilog Declaration".

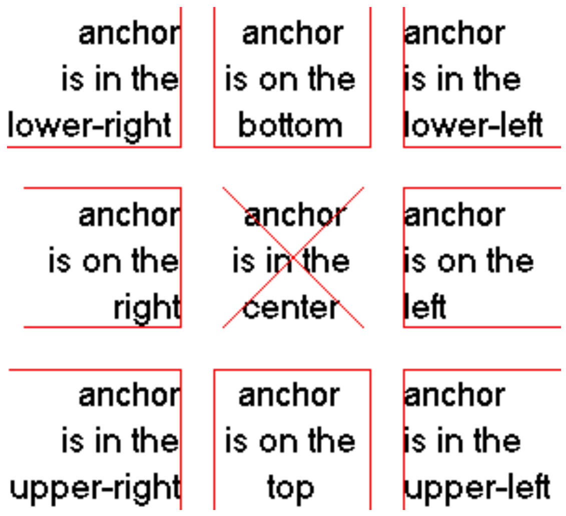
Essentially, then, every piece of text on the display is tied to some node or arc. By understanding the relationships between text and the attached objects, it becomes easy to manipulate that text.

6−8−2: Selecting Text

The only category of text that is not selectable is the text that is integral to a node's graphics (i.e. the Flip−Flop). For the rest, you can select and manipulate the text just as you would the object on which the text resides. (Note that port names on cell instances are not selectable: instead, select their export name inside of the cell definition.)

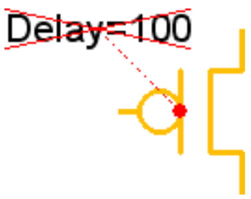
Certain types of text are not easily selectable. This is a feature that prevents accidental selection of unwanted text. For such hard−to−select text, the only way to select them is to use *special select* mode (see section 2−1−5). By default, the name of an unexpanded cell instance is hard−to−select. However, you can also request that names on nodes and arcs (annotation text) also be difficult to select by unchecking "Easy selection of annotation text" in the "Selection" preferences (in menu **File / Preferences...**, "General" section, "Selection" tab).

All text is attached to its node or arc at an *anchor−point*. This is the one point on the text that never moves, regardless of the size of the text. The highlighting of selected text varies according to the anchor−point. Typically, the highlighting consists of an "X" through the text. This indicates that the anchor−point is in the center. If a "U" is drawn in any of four orientations, it indicates that the anchor−point is on the side and that the text grows out of the opened end. If an "L" is drawn in any of four orientations, it indicates that the


anchor is in the lower-right
anchor is on the bottom
anchor is in the lower-left
anchor is on the right
anchor is in the center
anchor is on the left
anchor is in the upper-right
anchor is on the top
anchor is in the upper-left

anchor−point is in a corner.

Finally, the text may be drawn with an "X" but also have four lines that indicate a box at the object edge. This is centered text that clips to the size of its attached object (it is *boxed*).

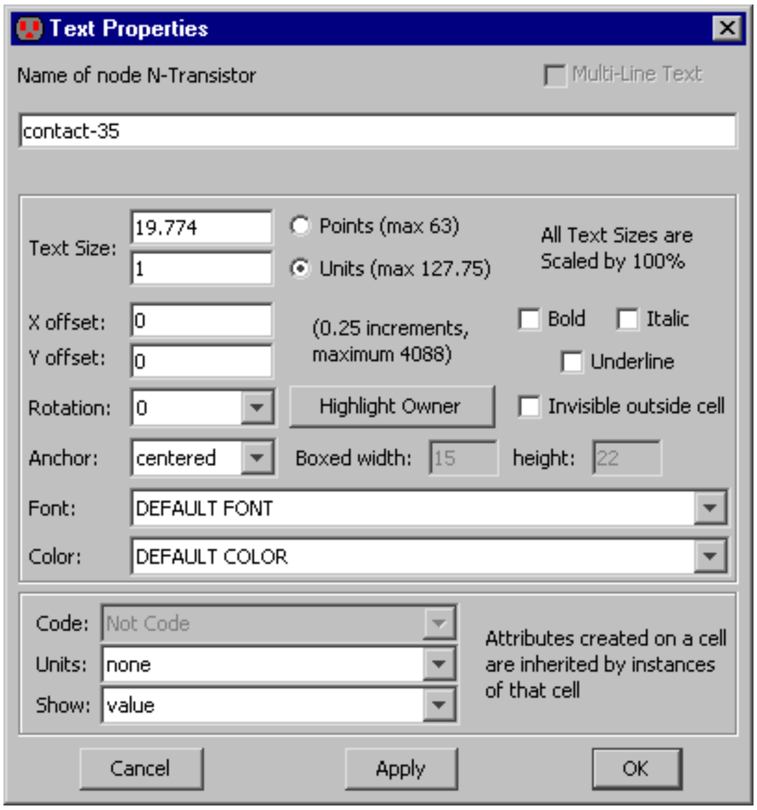


Note that text can be moved away from its attached node or arc. If this has been done, then selection of the text will also indicate the attached component by drawing a dashed line to it.

6−8−3: Modifying Text

Like nodes and arcs, text can be moved simply by clicking and dragging. It can be erased with the **Erase** command of the **Edit** menu (the Delete key).

To change text, double−click on it and type new values. To change other aspects of selected text, and use the **Object Properties...** command (in menu **Edit / Properties**). This dialog allows modification of the text, size, font, style, anchor−point, rotation, color, and even the offset of the anchor−point from the attached node or arc. Note that the offset is always relative to the center of the attached object. The size of text can be absolute (given in "points") or relative (given in units). The font of the text can be any the default font or any font installed on your system. The style of text can be any combination of Italic, Bold, or Underline. Text rotation can be in 90−degree increments only. You can set the units to any electrical type (capacitance, resistance, etc.) See section Section 7−2−2 for more on these units.


Text Properties

Name of node N-Transistor ☐ Multi-Line Text

contact-35

Text Size: 19.774 ○ Points (max 63)
1 ● Units (max 127.75)
All Text Sizes are Scaled by 100%

X offset: 0
Y offset: 0
(0.25 increments, maximum 4088)
☐ Bold ☐ Italic ☐ Underline

Rotation: 0 — Highlight Owner — ☐ Invisible outside cell

Anchor: centered — Boxed width: 15 — height: 22

Font: DEFAULT FONT

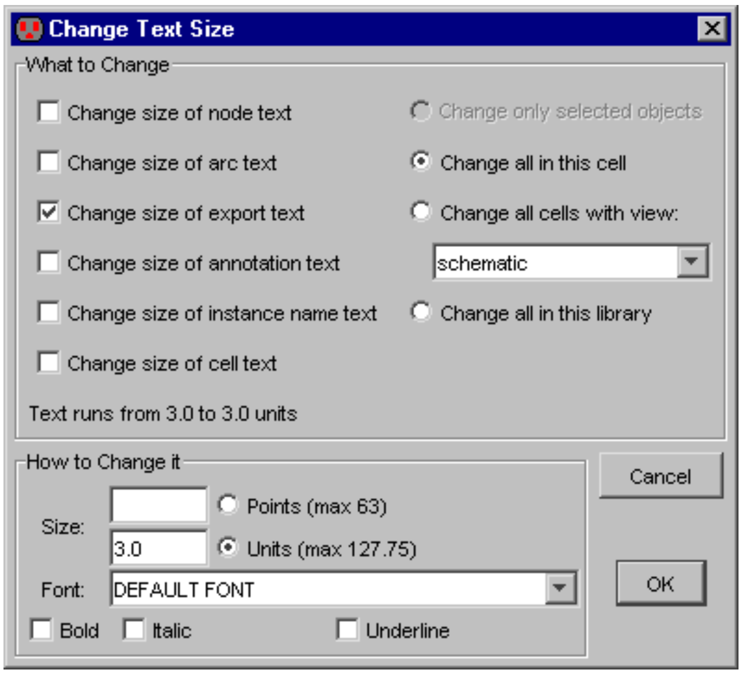
Color: DEFAULT COLOR

Code: Not Code
Units: none
Show: value
Attributes created on a cell are inherited by instances of that cell

Cancel Apply OK

The "Code" option allows the text to be code in an interpretive language, in which case, the evaluation of that code is displayed. You can choose to show the text value, the name of the piece of text, or both.

If the text contains more than 1 line, then you must check "Multi−Line Text" in order to edit it. You may then have to stretch the dialog in order to have a larger field for editing the text. The "Highlight Owner" button highlights the node or arc on which the text is attached . The checkbox "Invisible outside cell" requests that the text not be drawn when an instance of the cell is examined.


Change Text Size

What to Change

- [ ] Change size of node text
- [ ] Change size of arc text
- [✓] Change size of export text
- [ ] Change size of annotation text
- [ ] Change size of instance name text
- [ ] Change size of cell text

Change only selected objects [✓ Change all in this cell] Change all cells with view: schematic Change all in this library

Text runs from 3.0 to 3.0 units

How to Change it

Size: Points (max 63)

Size: 3.0 [✓ Units (max 127.75)]

Font: DEFAULT FONT

- [ ] Bold
- [ ] Italic
- [ ] Underline

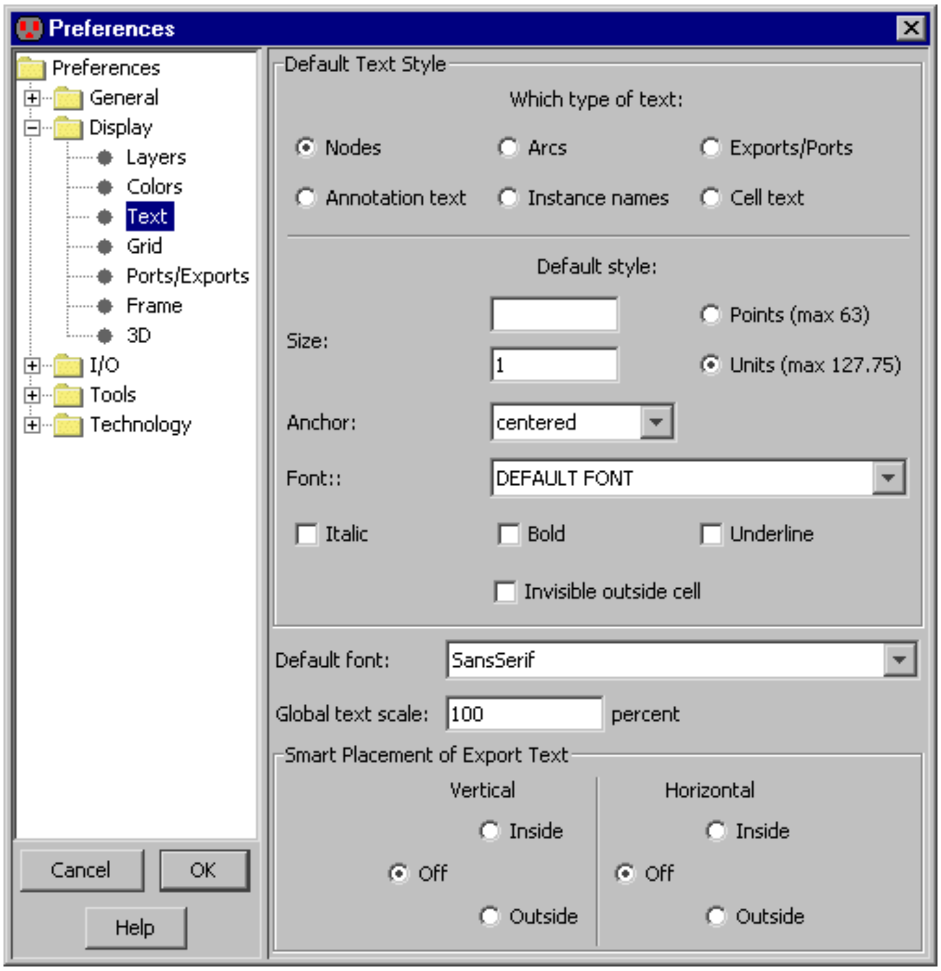
The **Change Text Size...** command (in menu **Edit / Text**) allows you to change the size, face, and style of any text object. You can choose which of the 6 classes of text you wish to change, and you can choose whether to make the changes only on selected objects, in the current cell, in all cells of a particular view, or everywhere.

6−8−4: Text Defaults

To change the default size and anchor−point of all new text, use the "Text" preferences (in menu **File / Preferences...**, "Display" section, "Text" tab). The top part of the dialog determines which types of text will be affected. The middle part lets you control how that type of text will appear in the future. You can set the size, anchor−point, font, and style.

Below the "Default Text Style" section is a popup that sets the default font for Electric. Below that is a setting for the "Global text scale". Normally, all text is drawn at 100% of its stated size. However, you can globally scale all text by typing a value other than 100 into this field. You can also use the **Increase All Text Size** and **Decrease All Text Size** commands (in menu **Edit / Text**) to change this value, and alter the size of all displayed text.

The bottom part of the dialog controls "smart placement" of text, which adjusts the grab point according to the environment of the text. This currently applies only to export names, which are placed relative to the arc connecting to the exported node. For example, if a node on the left end of a wire has an export, and the "Horizontal" placement is set to "Inside", then the export text will attach on the left side, causing the label to appear inside of the wire.



Default Text Style

Which type of text:

[✓ Nodes] Arcs Exports/Ports
Annotation text Instance names Cell text

Default style:

Size: Points (max 63)
Size: 1 [✓ Units (max 127.75)]
Anchor: centered
Font:: DEFAULT FONT
Italic Bold Underline
Invisible outside cell

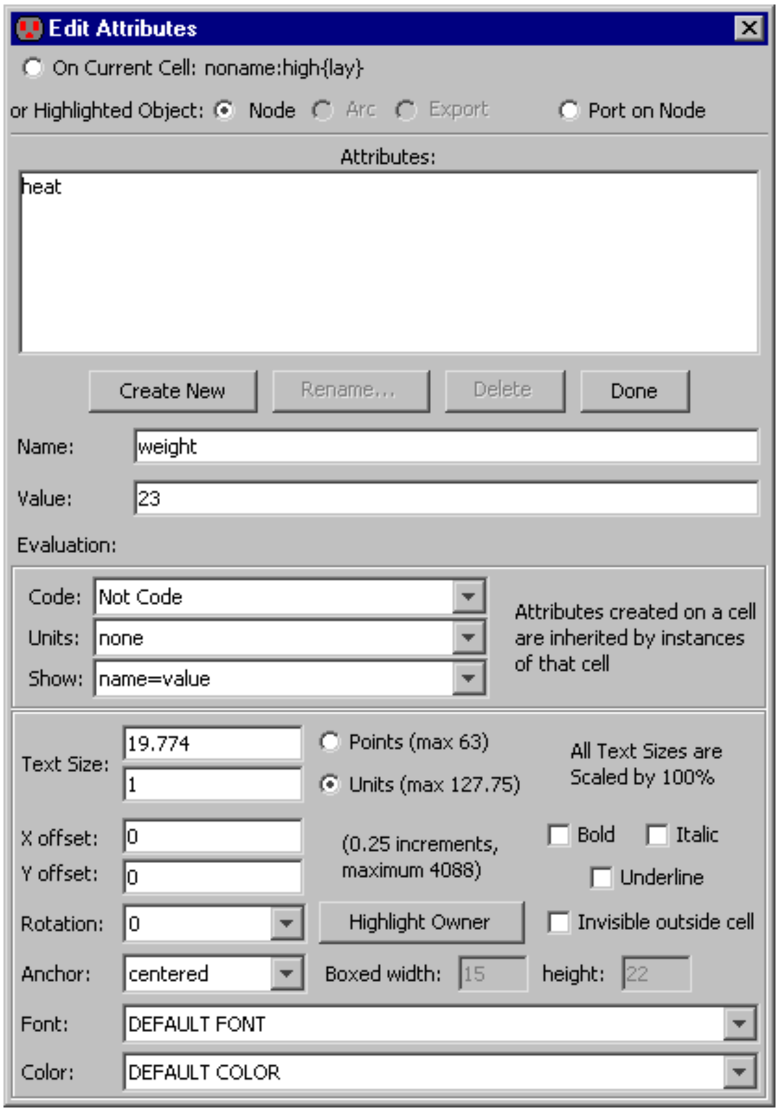
Default font: SansSerif
Global text scale: 100 percent

Smart Placement of Export Text

Vertical: Inside [✓ Off] Outside
Horizontal: Inside [✓ Off] Outside

6−8−5: Text Attributes

You can place arbitrary text attributes on nearly any part of the circuit by using the **Attribute Properties...** command (in menu **Edit / Properties**).


Edit Attributes

( ) On Current Cell: noname:high{lay}

or Highlighted Object: [✓ Node] ( ) Arc ( ) Export ( ) Port on Node

Attributes:

heat

Create New | Rename... | Delete | Done

Name: weight

Value: 23

Evaluation:

Code: Not Code

Units: none

Show: name=value

Attributes created on a cell are inherited by instances of that cell

Text Size: 19.774 ( ) Points (max 63)

1 [✓ Units (max 127.75)]

All Text Sizes are Scaled by 100%

X offset: 0

Y offset: 0

(0.25 increments, maximum 4088)

☐ Bold ☐ Italic ☐ Underline

Rotation: 0

Highlight Owner

☐ Invisible outside cell

Anchor: centered

Boxed width: 15 height: 22

Font: DEFAULT FONT

Color: DEFAULT COLOR

Attributes can be placed on these objects (selected at the top):

- The current cell.
- The currently highlighted node.
- The currently highlighted arc.
- The currently highlighted export.
- The currently highlighted port on the currently highlighted node.

The list of attributes is shown below that. You can create a new attribute by typing its name in the "Name:" field, its value in the "Value:" field and then clicking the "Create New" button. You can delete an attribute with the "Delete" button. An attribute's name can be changed with the "Rename" button.

Just below the name and value fields are a set of popups that control the attribute. The "Code" popup determines whether the attribute is code or pure data. This can be changed to one of the interpretive languages in Electric. When this happens, the attribute value is treated as code that is sent to that interpreter.

Then, the true value of the attribute is the evaluation of that code. For example, if the value of an attribute is "3+5" and the attribute is set to be Java code, then the Java interpreter will be invoked, and the attribute will actually be "8".

You can change the type of unit by using the "Units" popup (choices are capacitance, resistance, inductance, current, voltage, or distance). See Section 7−2−2 for more on these units.

You can control the way that an attribute is displayed in the circuit by selecting the appropriate entry in the "Show:" popup. You can request that various combinations of the attribute's name and value be displayed.

The bottom part of the dialog affects the appearance of the attribute, and is only relevant if the attribute is being shown. The size of the text can be specified in relative or absolute units. The anchor−point of the text can be set. The X and Y offset of the anchor−point from the attached object can be specified. The font of the text can be chosen. The style of the text can be any combination of Italic, Bold, or Underline. You can specify the rotation of the text, in 90−degree increments. You can even give the text a color. The "Invisible outside cell" requests that the attribute not be drawn when viewed farther up the hierarchy.

The "Done" button terminates this dialog. Note that there is no "Cancel" button: this dialog makes changes as they are entered.

Special Considerations

Attributes that are placed on cells or exports get *inherited* when the cell is instantiated. Each of these attributes is created on the instantiated node and port. It is often desirable for each inherited attribute to have unique names. If the value of a cell or export attribute has "++" in it, then the number before it will be incremented after inheritance. Similarly, a "−−" indicates that the number be decremented after inheritance. This allows an inherited attribute to be unique with each inheritance.

When there are too many visible attributes, the display can become cluttered. Use the "Layers" tab of the sidebar to control the text, (see Section 4−5 for more). Attributes on nodes are controlled by the "Node Text" checkbox; those on arcs with the "Arc Text" checkbox, etc.

6−8−6: Cell Parameters

When attributes are created on cells, they become parameters to that cell. Each instance of the cell has a copy of the attribute placed on the node, and the node's attribute can then be set independently of the cell's attribute. In such a situation, the node attribute's value is the "actual" parameter to the cell, and the cell attribute's value is the "formal", or default parameter value.

One example of the use of cell parameters is in the Spice primitives, where user−defined values (such as voltage) are communicated into the icon for generation in the Spice deck (see Section 9−4−3).

Another use of cell parameters is to parameterize the size of a transistor in a schematic (or to parameterize the scalable layout transistors in the MOSIS CMOS technology, see Section 7−4−2) The transistor width and length can be defined in terms of the parameter value, allowing a single cell to take on many different forms. By combining these parameters with the interpretive language facility, an arbitrary mathematical expression can be placed on the transistor which combines parameter values to form the exact transistor size .

Inside of the cell, the parameter is shown with its name and default value. To see the actual value from up the hierarchy, create a Java expression with the value "@PNAME" where PNAME is the parameter name. For example, if a cell has an attribute called "heat", you might place a piece of Annotation Text (with the

"Annotation Text" command under the "Misc" entry in the component menu) and name it "@heat". Make sure to set its Code to Java. When first defined, the parameter has no actual value, and so appears as "heat not found" . If an instance is created and its "heat" attribute is set to 7, then descending into that cell will show "7".

If a parameter is added to a cell, existing instances of that cell may not get properly updated. To do this, use the **Update Attributes Inheritance on Node** command (in menu **Edit / Properties**). To do this everywhere, use the **Update Attributes Inheritance all Libraries** command.

Parameter Text

Parameters on cells are not tied to any node or arc. Instead, they float freely inside of the cell. You can select the text and drag it to any location in the cell.

Parameters on instances of cells are placed at the same location as they appear inside of the cell. In schematics, the location of a parameter on an icon is determined by the location of that parameter on the sample icon, inside of the schematic cell.

If you do not wish to see a parameter's text on any instance, select the parameter text inside of the cell, use the **Object Properties...** command (in menu **Edit / Properties**), and check "Invisible outside cell".

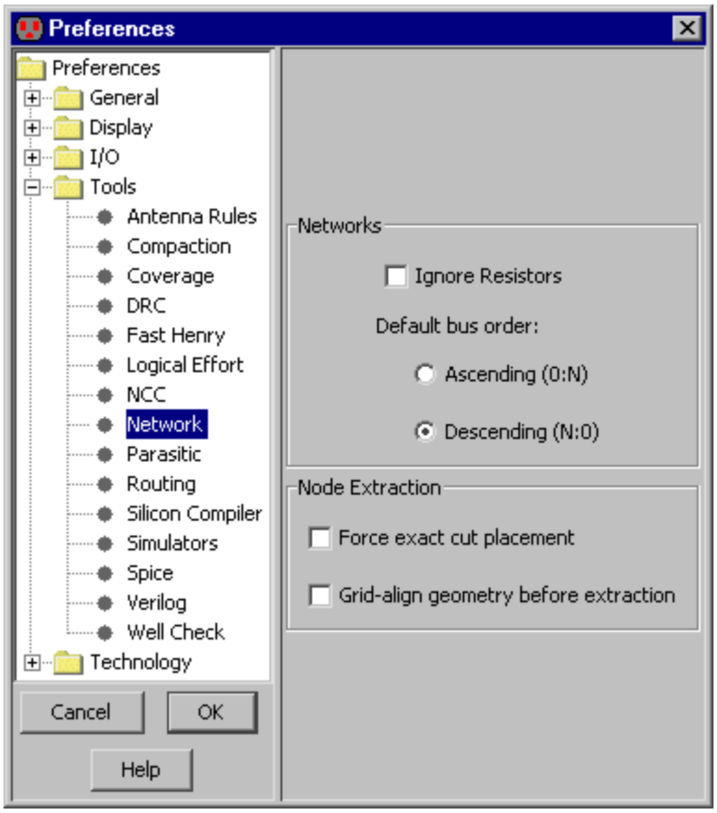
You can disable the display of parameters on cell instances by selecting them and using the **Hide All Attributes on Node** command (in menu **Edit / Properties**). You can force all parameters to be displayed on a cell instance by using the **See All Attributes on Node** command. To restore default parameter visibility on a cell instance, use the **Default Attribute Visibility** command.

6−9−1: Introduction to Networks

A collection of electrically connected components defines a *network*. Networks may span many arcs, or they may reside on only a single export on a single node. Because networks are stored in the Electric database, they can be immediately accessed when needed.

Whenever a port on a node is selected, the highlighting indicates the entire network that is connected to that port. Another way to see an entire network is to use the **Show Network** command (in menu **Tool / Network**). This will highlight all arcs on the currently selected networks. Repeated use of this command causes the network to be highlighted at successively lower levels of the hierarchy.

If the design is very dense, you can select one or more networks by name with the **Select Object...** command (in menu **Edit / Selection**).



Networks
☐ Ignore Resistors
Default bus order:
○ Ascending (0:N)
● Descending (N:0)

Node Extraction
☐ Force exact cut placement
☐ Grid-align geometry before extraction

The Resistor can be treated as a connecting or nonconnecting node. By default, it does not connect the networks on its two ends, so identification of the extent of a network ends at the resistor. To ignore resistors and treat them as wires, use the "Network" preferences (in menu **File / Preferences...**, "Tools" section, "Network" tab), and check "Ignore Resistors". Then highlighted networks will pass through them. See section Section 7−5−1 for more on resistors.

The other controls in the "Network" Preferences are discussed elsewhere. For an explanation of "Default bus order", see Section 6−9−3). For an explanation of the entries in the "Node Extraction" section, see Section 9−10−2).

There are many commands in menu **Tool / Network** that can be used to get information about the networks in a cell:

- **List Networks** shows a list of the networks in the current cell.
- **List Exports on Network** lists all export names on the currently highlighted network. This list contains the names of exports at all levels of the hierarchy, above and below the current cell. The facility is useful if, for example, you have propagated clock lines throughout the circuit and wish to make sure that all of the export names on this network have some variant of the name "phi". By quickly examining this list, you can see all of the names that have been used on the network, throughout the hierarchy.
- **List Exports below Network** lists all export names on the currently highlighted network. This list is similar to the one generated by **List Exports on Network** except that it works only on cells below the current one.
- **List Connections on Network** lists all nodes in the current cell that are connected to the current network. This list includes only those nodes at the ends of the net, not the pin or contact nodes used inside of the network. The command is useful if you are at one end of a wire and want to check to see what is at the other end.
- **List Geometry on Network** lists all geometry in the current cell that is connected to the current network. This reports the area and perimeter of all attached layers.
- **List Total Wire Lengths on All Networks** lists the lengths of all networks in the current cell.

6−9−2: Naming Networks

Network names are derived from export names and arcs that are named in a cell. The name given to an export becomes the network name for all arcs connected to that export. Similarly, the name given to an arc (by setting the name field in the **Object Properties...** dialog) becomes the name of the network for all connected arcs. You can rename a network by changing the name of a connected export or arc.

Two phenomena can occur in network naming: a network can be *multiply named*, and it can *span disjoint circuitry*. A network has multiple names when two or more connected arcs or exports are named with different names. For example, if you make an export on a contact node and call it "clock", then you select an arc connected to that contact node and name it "sig", the circuitry will be on the network "clock/sig." Thus, both names now apply to the same network.

The other phenomenon of network naming is that a single network can include unconnected parts of the circuit. This happens when arcs in unconnected parts of the circuit are given the same name. This causes the two arcs to be implicitly joined into one network. Because this network naming phenomena is most commonly used in schematics, the unification of like−named networks only happens in cells with the "schematic" view.

6−9−3: Bus Naming

The Bus arc of the Schematics technology is a special arc that can carry multiple signals (see Section 7−5−1). When giving a network name to Bus arcs, it is possible to specify complex bus names.

- **Lists** Bus names can be lists (for example, "clock,in1,out" which aggregates 3 signals into a 3−wide bus) .
- **Simple arrays** Bus names can be arrays (for example, "A[0:7]" which defines an 8−wide bus).
- **Array index lists and ranges** Arrayed bus names can have lists of values (separated by commas) or ranges of values (using the colon). For example, the bus "b[0],c[3,5],d[1:2],e[8:6]" is an 8−wide bus with signals in this order: b[0], c[3], c[5], d[1], d[2], e[8], e[7], e[6].
- **Multidimensional array indices** Arrays can be multiply indexed (for example "b[1:2][100,102]" defines a bus with 4 entries: b[1][100], b[1][102], b[2][100], and b[2][102]). You can have any number of dimensions in an array.
- **Symbolic array indices** It is possible to use symbolic indices in bus naming (for example, the bus "r[x,y]" defines a 2−wide bus with the signals r[x] and r[y]).

When a bus is unnamed, the system determines its width from the ports that it connects. Some tools (such as simulation netlisters) need to name everything, and so use automatically−generated names. When this happens, the system must choose whether to number the bus ascending or descending. To resolve this issue, use the "Network" preferences (in menu **File / Preferences...**, "Tools" section, "Network" tab), and choose "Ascending" or "Descending".

Individual wires that connect to a bus must be named with names from that bus. As an aid in obtaining individual signals from a bus, the **Rip Bus** command (in menu **Edit / Arc**) will automatically create such wires for the selected bus arc.

Besides using array names on busses, you can also give array names to schematic nodes. Netlisters will create multiple copies of that node, named with the individual elements of the array.

6−9−4: Power and Ground

Identification of a power network is done by finding:

- a Power node from the Schematic technology;
- an export in the current cell that has the "power" characteristic;
- an export in the current cell that begins with the letters "vdd", "vcc", "pwr", or "power";
- a port on a component in the current cell that has either of the above two properties.

Ground networks use the same rules, except that the acceptable port names begin with "vss", "gnd", or "ground".

All supply networks defined with the Power and Ground nodes of the Schematic technology are combined into one network. This means, for example, that two arcs, each connected to a separate Ground node, appear on the same network regardless of their actual connectivity in the circuit.

As a debugging aid for power and ground networks, the command **Show Power and Ground** (in menu **Tool / Network**) shows the entire power and ground network. The **Validate Power and Ground** command checks all power and ground networks in the circuit. Any power or ground networks that are named according to the prefixes listed above must have the proper characteristics. If, for example, a power network is called "gnd007", then it will be flagged by this command.

6−9−5: Global Networks

When wiring an IC layout, the only way to get a signal from one point to another is to physically place the wires. Signals that span a large circuit, such as power and ground, must be carefully wired together at each level of the hierarchy.

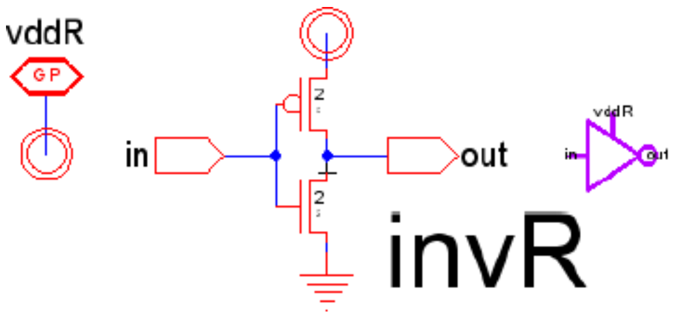
In schematics, however, it is often the case that a signal is used commonly without explicitly being wired or exported. Examples of such signals are power, ground, clocks, etc. The power and ground signals can be established in any schematic with the use of the Power and Ground nodes. To create another such signal, use the Global node of the schematics technology (see Section 7−5−1).

The Global node is diamond−shaped, and it has a name and characteristic similar to exports (input, output, etc.) All signals with the same global name are considered to be connected when netlisting occurs. Thus, the Global symbol can be used to route clock signals, as well as to define multiple power and ground rails. Note that with multiple power and ground rails, only one of them is the true "power and ground" as defined by the Power and Ground symbols. All others, declared with Global nodes, are not true power and ground signals, but are simply globals.

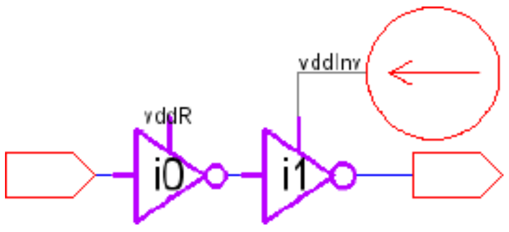
Global Partitioning

It is sometimes the case that the designer wishes to isolate a global signal and wire it differently. For example, a schematic cell may be defined with power and ground symbols, connecting it to the global power and ground. But a particular instance of the schematic may need to be wired to alternate power and ground rails, for example "dirty power". Another example of rewiring happens when you want to test a specific instance of a cell, and you need to connect its globals differently for the purposes of simulation.

The solution is to place a "Global Partition" node inside of the schematic (see Section 7−5−1). This symbol acts like an "offpage" symbol: it is wired to something inside of the cell (a global signal) and it is also exported to the outside world.



In this example, the schematic has power and ground signals, but the power signal is also connected to a Global Partition node and exported (as "vddR"). The icon has an extra connection for this power tap. In normal use, the extra connections created by the Global Partition nodes do not need to be wired up, because they connect to globals, and their connectivity is understood. If, however, the extra exports *are* wired, it means that the signal inside of the cell is disconnected from the global, and connected instead to that wire.

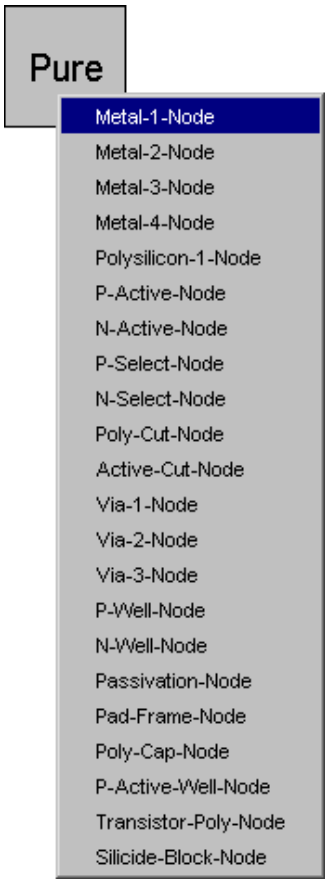


In the example here, two "invR" icons are placed, but one of them has its "vddR" connection wired (to a different power source). The subcircuit for the rightmost icon will not use the global power signal, but will instead use the attached signal, "vddInv".

Chapter 6: Advanced Editing

6−10−1: Introduction to Outlines

For some primitive nodes, it is not enough to rotate, mirror, and scale. These primitives can to be augmented with an *outline*, which is a polygonal description.



There are quite a few primitive nodes that make use of outline information. The MOS transistors use the outline to define the gate path in serpentine configurations (see Section 7−4−1). The Artwork technology has nodes that use outline information: Opened−Solid−Polygon, Opened−Dotted−Polygon, Opened−Dashed−Polygon, Opened−Thicker−Polygon, Closed−Polygon, Filled−Polygon, and Spline (see Section 7−6−1).

For arbitrary shapes on arbitrary layers, use the *pure−layer* nodes in the IC layout technologies. The pure−layer nodes are found under the "Pure" entry in the component menu. For example, the node called "Metal−1−Node" in the CMOS technologies looks like a rectangle of the Metal−1, until you add outline information. With an outline, this node can take any shape.

Because pure−layer nodes are unusual, it is useful to be able to identify them. Use the **Show Pure Layer Nodes** command (in menu **Edit / Cleanup Cell**) to highlight all of them in the current cell.

6−10−2: Manipulating Outlines

To manipulate outline information on the currently highlighted node, use "Outline Edit" mode (click on the icon in the tool bar or use the **Toggle Outline Edit** command, in menu **Edit / Modes / Edit**).

In this mode, there is always a "current point", identified with an "X" over it. To further identify this point, the lines coming into and out of the point have arrows on them indicating the direction of the outline.

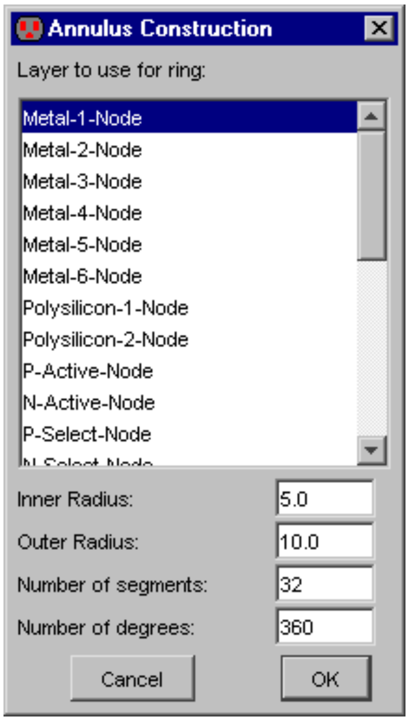
In outline edit mode, the *left* button is used to select and move a point on the outline, and the *right* button adds a new point after the selected one.

Besides selecting points with the mouse, you can also step through the points of the outline with the "." key (next outline point) and "," key (previous outline point). These keys are under the ">" and "keys, so you can think of them as the "next point" (>) and "previous point" (<) commands.

The **Erase** command (in menu **Edit**) deletes the current outline point (so does the Delete key).

When done editing the outline, switch to standard selection mode (the **Click/Zoom/Wire** command, in menu **Edit / Modes / Edit**).

6−10−3: Special Outline Generation


Annulus Construction

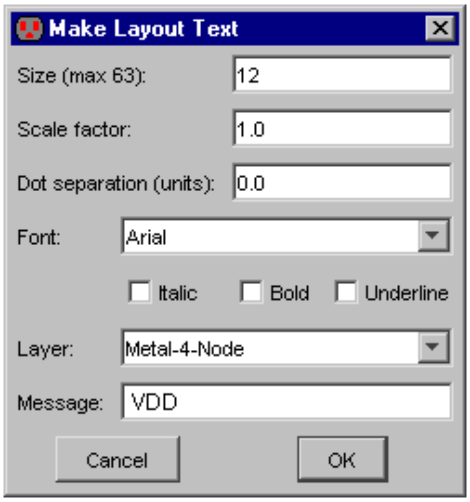
Layer to use for ring:

Metal-1-Node
Metal-2-Node
Metal-3-Node
Metal-4-Node
Metal-5-Node
Metal-6-Node
Polysilicon-1-Node
Polysilicon-2-Node
P-Active-Node
N-Active-Node
P-Select-Node

Inner Radius:	5.0
Outer Radius:	10.0
Number of segments:	32
Number of degrees:	360

To generate a doughnut shaped outline, use the "Annular Ring..." command under the "Misc" entry in the component menu. This dialog prompts for a layer to use and an inner and outer radius for the annulus. By default, it is made as a full circle (360 degrees), but this can also be changed. Finally, the number of line segments used in the construction can be set, allowing for smoother or coarser shapes.

To generate text−shaped outlines, use the "Layout Text..." command under the "Misc" entry in the component menu. This dialog prompts for text and a layer to use as well as the size, scale, font, and style. A nonzero dot separation causes each pixel of the text to be placed separately (some design rules need this).


Make Layout Text

Size (max 63):	12
Scale factor:	1.0
Dot separation (units):	0.0
Font:	Arial
	☐ Italic ☐ Bold ☐ Underline
Layer:	Metal-4-Node
Message:	VDD

6−11: Interpretive Languages

Electric has the Bean Shell built into it. This enables you to load Java scripts that access the Electric database.

The Bean Shell is not part of the default Electric distribution. You must add it as a "plug in" (see Section 1−5 for more on plug−ins).

To run a script, use the **Run Java Bean Shell Script** command (in menu **Tool / Languages**). Here is an example script that searches the current cell for exports starting with "a".

```
import com.sun.electric.database.hierarchy.Cell;
import com.sun.electric.tool.user.ui.WindowFrame;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
// read library with test setup
Cell lay = WindowFrame.getCurrentCell();
// find all exports
com.sun.electric.database.hierarchy.Export e;
List aList = new ArrayList();
for(Iterator it = lay.getPorts(); it.hasNext(); )
{
    e = (com.sun.electric.database.hierarchy.Export)it.next();
    if (e.getName().startsWith("a")) aList.add(e);
}
String aOut = "Exports that start with 'a':";
for(Iterator it = aList.iterator(); it.hasNext(); )
{
    e = (com.sun.electric.database.hierarchy.Export)it.next();
    aOut += " " + e.getName();
}
System.out.println(aOut);
```

Notice that Electric's "Export" object must be a fully−qualified name, because the name "Export" is used for other reasons in the Bean Shell.

For more information about accessing the interals of Electric, read the Javadoc in the source code.

Chapter 6: Advanced Editing

6−12: Project Management

The project management system in Electric allows multiple users to work together on the design of a circuit. This is accomplished by having a *repository* in a shared location, and local libraries in each user's disk area. Users work on cells by checking them out of the repository, making changes, and then checking them back in. The project management system ensures that only one user can access a cell at a time. In addition, it also applies its understanding of the circuit hierarchy to inform users of potential inconsistencies that may arise.

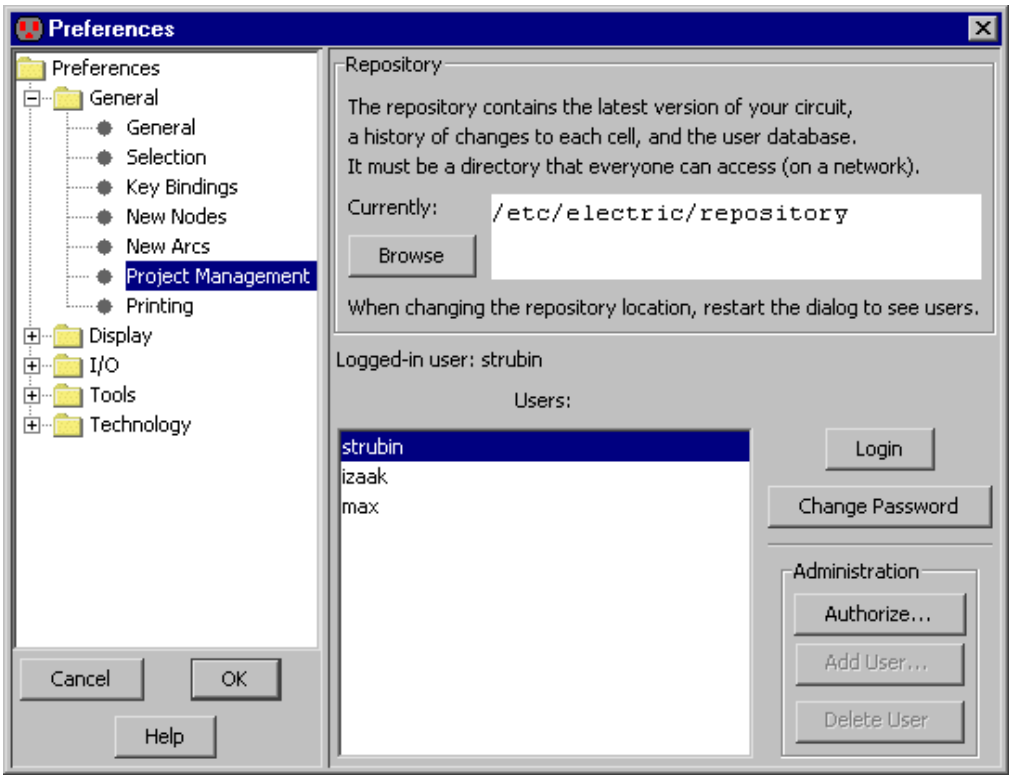
The project management system uses the full power of cells to accomplish its task. It handles design history by creating a new version of a cell each time it is checked out of the repository. The user's library contains only the most recent version of each cell, taken from the repository. When a user updates their library from the repository, newer versions are brought in and substituted for older versions. Unless the user specifically asks for an older version, it is removed from their library.

Because the project management system uses versions to manage design progress, users are discouraged from managing versions explicitly. Thus, the command **New Version of Current Cell** (in menu **Cell**) is not allowed. Also, it is not appropriate for a user to use two different versions of a cell explicitly, because they are considered to be part of a single cell's history.

All commands to the project management system can be found under the **Project Management** command (in menu **File**). Subcommands exist there for checking cells in and out, updating local libraries from the repository, and more. Many project management functions are also available in context menus in the cell explorer.

Setting Up Project Management

The first step needed to use the project management system is to choose a location for the repository. This must be a shared location that each user can access (read and write). Use the "Project Management" preferences, in menu **File / Preferences...**, "General" section, "Project Management" tab. Each user must do this and set the same location so that they can share the repository.



The repository contains the latest version of your circuit,
a history of changes to each cell, and the user database.
It must be a directory that everyone can access (on a network).

Currently: /etc/electric/repository

When changing the repository location, restart the dialog to see users.

Logged-in user: strubin

Users:

strubin
izaak
max

After the repository has been set, libraries can be entered into it. Use the **Add Current Library To Repository** command to place your library in the repository. Note that a library that has been entered into the repository is also tagged with information about the repository location, as well as the state of the cells (checked−in or checked−out). Therefore, you should save your library after entering it into the repository.

Other users can obtain a copy of your library directly from the repository by using the **Get Library From Repository...** command.

Individual users can now begin to work on the library. Before checking cells out of a library, it is necessary to create a "user" account in the project management system. The "Project Management" preferences shows a list of users. Although each user must "login" to the project management system with their own password, Electric remembers the logged−in user between sessions, so it is not necessary to do this more than once. Adding new users can be done in the "Administration" section of the dialog. Note that to add or delete users, the administrator must first click "Authorize..." and provide the appropriate password. This password is provided elsewhere in the manual.

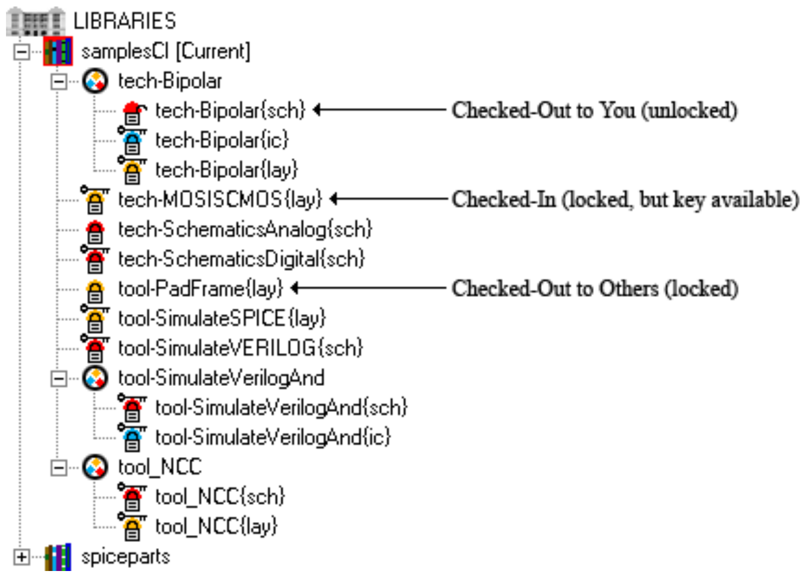
Checking Cells In and Out

When a cell is not checked out, you cannot make changes to it. Any change is immediately undone by the project management system. This means that a change which affects unchecked−out cells, higher up the hierarchy, will also be disallowed.

To check−out the current cell, use the **Check Out This Cell** command. If there are related cells (hierarchically above or below this) that are already checked−out to other users, you will be given warnings about potential conflicts that may arise.

To check the current cell back in, use the **Check In This Cell...** command. You will be prompted for a documentation message about the change. No further changes will be allowed to the cell. Note that when checking−in a cell, other cells above and below this in the hierarchy will also be checked−in. This is because changes affect other cells in the hierarchy, and so consistent pieces of the hierarchy must be updated at the same time.

The cell explorer shows the state of cells that are under project management control (see Section 4−8). Locks are drawn over cells to indicate their state (checked−in, checked−out to you, or checked−out to others). You can also access many of the project management commands by selecting cells in the explorer and using context menu commands.


LIBRARIES
samplesCI [Current]
tech-Bipolar
tech-Bipolar{sch}
Checked-Out to You (unlocked)
tech-Bipolar{ic}
tech-Bipolar{lay}
tech-MOSISCMOS{lay}
Checked-In (locked, but key available)
tech-SchematicsAnalog{sch}
tech-SchematicsDigital{sch}
tool-PadFrame{lay}
Checked-Out to Others (locked)
tool-SimulateSPICE{lay}
tool-SimulateVERILOG{sch}
tool-SimulateVerilogAnd
tool-SimulateVerilogAnd{sch}
tool-SimulateVerilogAnd{ic}
tool_NCC
tool_NCC{sch}
tool_NCC{lay}
spiceparts

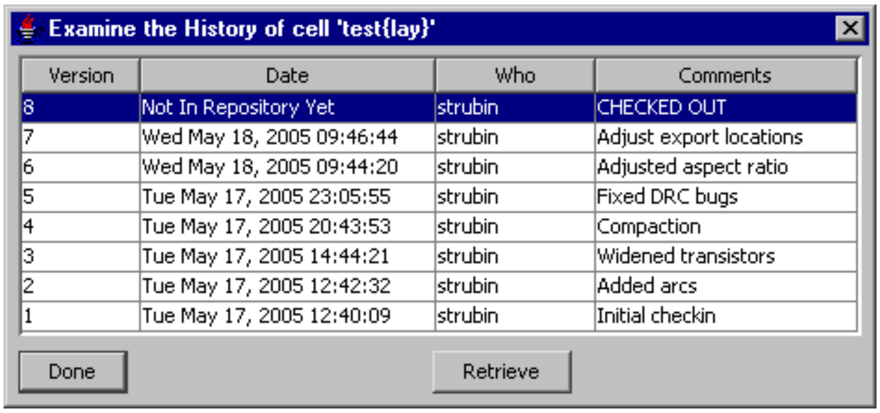
To update your library so that it contains the most recent version of every cell, use the **Update** command. This will retrieve the newest version of every cell in every library that is being managed. You will be given a list of cells that were replaced.

Advanced Commands

If, after a cell has been checked−out, you change your mind and do not wish to make changes, use the **Cancel Check−Out** command (or use the "Cancel Check−Out" context menu when clicking on a cell name in the cell explorer). This will destroy any changes made to the cell since it was checked−out and revert the cell to its state when it was checked−in.

If, in the course of design, a new cell is created, it must be added to the repository so that others can share it. Use the **Add This Cell** command to include the cell in the repository. Similarly, if a cell is to be deleted, use the **Remove This Cell** command to delete it from the repository.

To examine the history of changes to a cell, use the **Show History of This Cell...** command (or use the "Show History of This Cell..." context menu when clicking on a cell name in the cell explorer). Besides showing the history of changes, you can use this dialog to retrieve an earlier version of the cell.



Examine the History of cell 'test{lay}'

Version	Date	Who	Comments
8	Not In Repository Yet	strubin	CHECKED OUT
7	Wed May 18, 2005 09:46:44	strubin	Adjust export locations
6	Wed May 18, 2005 09:44:20	strubin	Adjusted aspect ratio
5	Tue May 17, 2005 23:05:55	strubin	Fixed DRC bugs
4	Tue May 17, 2005 20:43:53	strubin	Compaction
3	Tue May 17, 2005 14:44:21	strubin	Widened transistors
2	Tue May 17, 2005 12:42:32	strubin	Added arcs
1	Tue May 17, 2005 12:40:09	strubin	Initial checkin

Under the Hood

The project management system makes use of version information on all cells to control cell changes. When a cell is checked−out, a new version is made in your local library, and the old version is deleted. All instances of the old version are switched to the new version. The old version remains in the repository. When the cell is checked−in, that new version also goes into the repository. When updates are done, newer versions are obtained from the repository, and appropriate substitutions are performed.

It is assumed that anyone who has read all the way to the end of this manual page must be quite serious about the project management system. Such a person is probably an administrator, and therefore deserves to know what the administration password is for adding and deleting users. The password is just the letter "e". For increased security, edit the code and change this to something more secure.

6−12: Emergencies

Electric uses separate Java threads for all activities. Because of this, if the system encounters an error, it aborts the thread but the main program continues to run.

If a thread crashes and leaves a Job running, then you will not be able to issue other commands, because their Jobs will be queued behind the stuck one (see Section 4−8 for more viewing Jobs). Even the **Quit** command is a job, and so it cannot run. To solve this problem, use the **Force Quit (and Save)** command (in menu **File**).

If you suspect that the database is corrupt, use the subcommands of the **Check Libraries** command (in menu **File**). The **Check** command examines the database but does not fix errors. The **Repair** command checks and repairs the database (if it can).

The networks may also need to be renumbered. Do this with **Redo Network Numbering** command (in menu **Tool / Network**).

Chapter 7: Technologies

7−1−1: Technologies

A *technology* is an environment in which design is done. Technologies can be layout specific, for example MOSIS CMOS, or they can be abstract, for example Schematics and Artwork. There are multiple CMOS variations to handle popular design rules such as MOSIS, submicron, etc.

Each technology consists of a set of *primitive nodes* and *arcs*. These, in turn, are constructed from one or more *layers*. Each technology also includes information necessary to do design, such as design rules, connectivity rules, simulation information, etc.

The primitive nodes in a technology come in three styles: *pins*, *components*, and *pure−layer nodes*. The pins are used to join arcs, so there is one pin for every arc in the technology. The components are the basic nodes used in design: contacts, transistors, etc. Finally, the pure−layer nodes are used for geometric manipulation (see Section 6−10−1). There is one pure−layer node for every layer in the technology.

The component menu in the side bar (on the left side of the editing window) shows arcs on the left (the menu entries with red border), pin nodes in the center column (these appear as boxes with a cross inside), and components on the right (the more complex layer combinations). The pure−layer nodes are available under the entry labeled "Pure".

Electric has a "sample" library built into it that illustrates many features. To access it, use the **Load Library** command (in menu **Help / Samples**). The table below lists the cells in that library which illustrate different technologies:

Technology	**Description**	**Sample Cell**
mocmos	MOSIS CMOS rules	tech−MOSISCMOS{lay}
bipolar	Simple bipolar technology	tech−Bipolar{lay}
schematics	Digital schematics layout	tech−SchematicsDigital{sch}
schematics	Analog schematics layout	tech−SchematicsAnalog{sch}
artwork	Graphical design	tech−Artwork
pcb	Printed Circuit Boards	tech−PCB{sch}
nmos	n−Channel MOS rules	tech−nMOS{lay}
rcmos	Round CMOS rules	tech−RoundCMOS{lay}
efido	Digital Filter technology	tech−DigitalFilter
gem	Temporal Logic specification	tech−Gem

7−1−2: Controlling Technologies

Electric has the concept of a *current technology* which is shown in the status bar. This technology affects many things, including the selection of nodes and arcs in the component menu. There are a number of ways to affect the current technology, both manual and automatic.

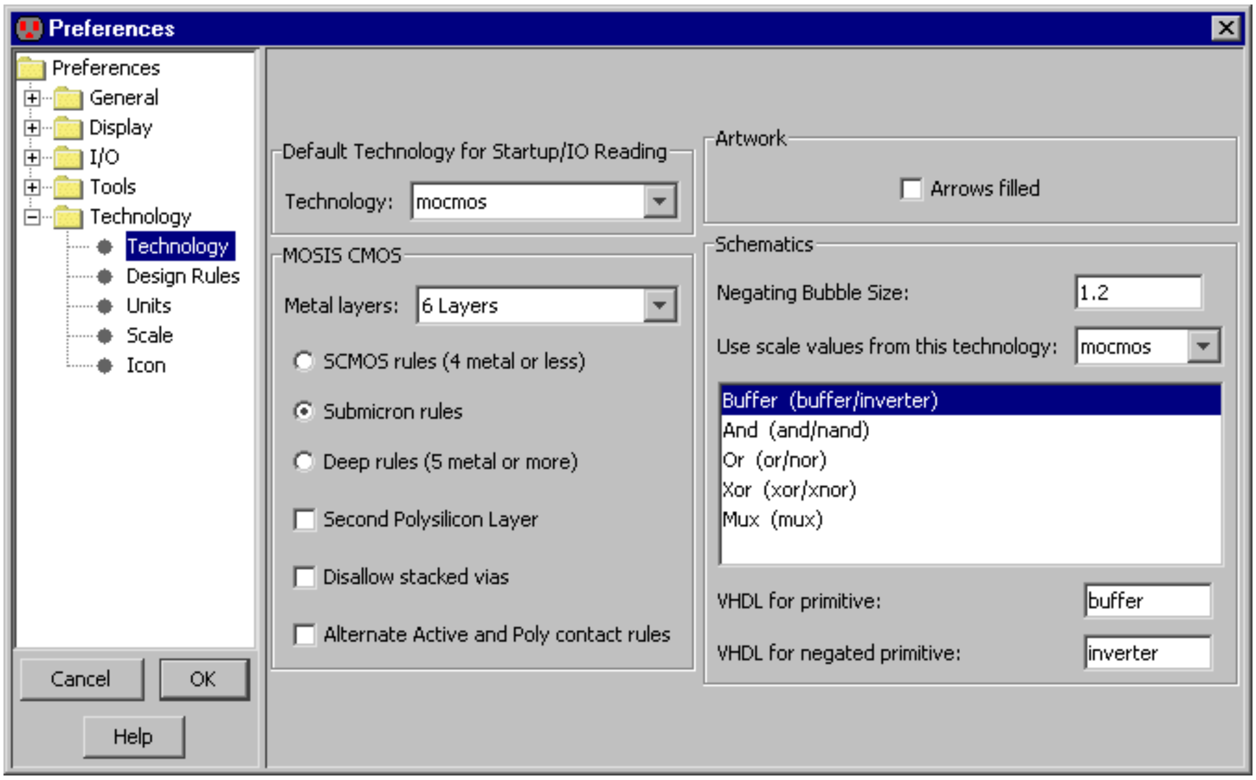
You can change the current technology by selecting it from the popup at the top of the side bar (either the "Components" or "Layers" tab). Electric automatically switches the current technology to match the cell being edited. If there are multiple cells being edited from different technologies, this switching can become annoying. To disable automatic technology switching, use the "New Nodes" preferences (in menu **File / Preferences...**, "General" section, "New Nodes" tab), and uncheck "Switch technology to match current cell".

To see a list of primitive nodes and arcs in the current technology, use the **Describe this Technology** command (in menu **Edit / Technology Specific**). To see a detailled description of the current technology, use the **Document Current Technology** command.

The "Technology" preferences (in menu **File / Preferences...**, "Technology" section, "Technology" tab) lets you control many general and specific technology settings.

You can change the default technology with the popup in the upper−left of this dialog. This default technology is the used when Electric first begins. It is also used when reading old libraries that are missing some technology information.

Besides remembering a startup technology, Electric also remembers a default "layout" technology. This is a technology with real geometry (an integrated circuit technology, not a schematics or artwork technology) The default layout technology is used to give further information about schematics components . It is listed in the "Schematics" section under "Use scale values from this technology".



Default Technology for Startup/IO Reading
Technology: mocmos

MOSIS CMOS
Metal layers: 6 Layers
SCMOS rules (4 metal or less) [✓ Submicron rules] Deep rules (5 metal or more)
☐ Second Polysilicon Layer
☐ Disallow stacked vias
☐ Alternate Active and Poly contact rules

Artwork
☐ Arrows filled

Schematics
Negating Bubble Size: 1.2
Use scale values from this technology: mocmos
Buffer (buffer/inverter)
And (and/nand)
Or (or/nor)
Xor (xor/xnor)
Mux (mux)
VHDL for primitive: buffer
VHDL for negated primitive: inverter

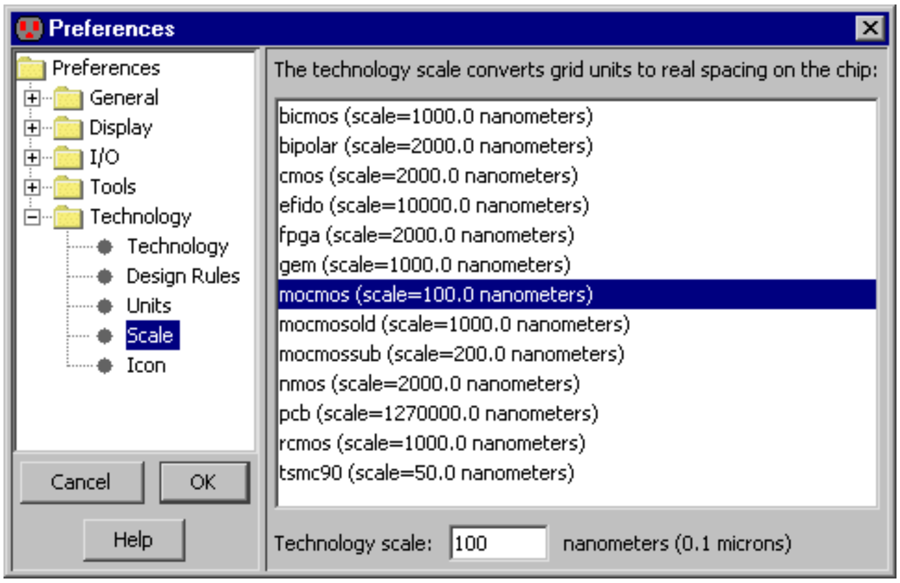
Some technologies have settable options that further customize them. The "Technology" preferences dialog lets you control these options. More information about this dialog is available from the individual technology sections on MOSIS CMOS (Section 7−4−2), Schematics (Section 7−5−1 and Section 3−11−2), and Artwork (Section 7−6−1).

7−2−1: Scale

Electric represents all distances in dimensionless units. A transistor that is 2 x 3 in size is actually stored in memory as 2 x 3. To convert these units to real distances, each technology has a *scale*, measured in nanometers (billionths of a meter). The scale of a technology is shown in the status area after the technology's name.

For example, if the scale for the MOSIS CMOS ("mocmos") technology is 200 nanometers, then a 2 x 3 transistor is actually 400 x 600 nanometers (or 0.4 x 0.6 microns).

To set the scale, use the "Scale" preferences (in menu **File / Preferences...**, "Technology" section, "Scale" tab).



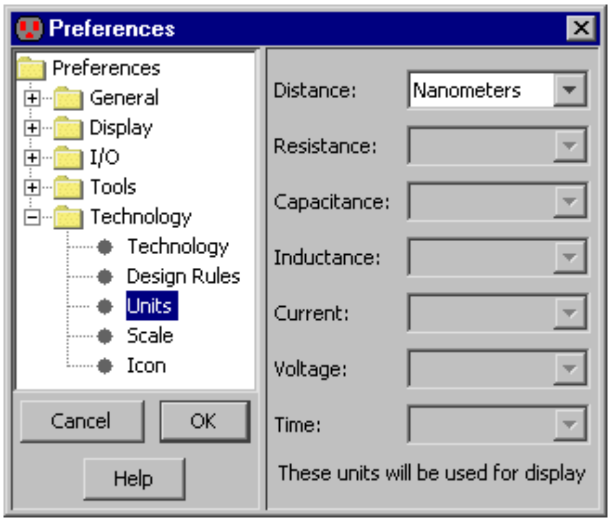
The technology scale converts grid units to real spacing on the chip:

bicmos (scale=1000.0 nanometers)
bipolar (scale=2000.0 nanometers)
cmos (scale=2000.0 nanometers)
efido (scale=10000.0 nanometers)
fpga (scale=2000.0 nanometers)
gem (scale=1000.0 nanometers)
mocmos (scale=100.0 nanometers)
mocmosold (scale=1000.0 nanometers)
mocmossub (scale=200.0 nanometers)
nmos (scale=2000.0 nanometers)
pcb (scale=1270000.0 nanometers)
rcmos (scale=1000.0 nanometers)
tsmc90 (scale=50.0 nanometers)

Technology scale: 100 nanometers (0.1 microns)

Scale only applies to integrated−circuit layout technologies. There is no scale for Schematics, Artwork, and other nonlayout technologies.

7−2−2: Units



Distance: Nanometers
Resistance:
Capacitance:
Inductance:
Current:
Voltage:
Time:
These units will be used for display

Although distances are described in dimensionless units, they must be expressed with real units when converted to the real world. You can choose which unit should be used with the "Units" preferences (in menu **File / Preferences...**, "Technology" section, "Units" tab).

Much of this dialog is currently unavailable, because the system does not use this information.

7−3−1: I/O Specifications

Electric is able to read and write circuits in a number of different formats. This is done with the **Import** and the **Export** commands (in menu **File**). See Section 3−9−2 for more on Import; see Section 3−9−3 for more on Export.

To properly control translation, use the many preferences dialogs for the different file types (in menu **File / Preferences...**, "I/O" section).

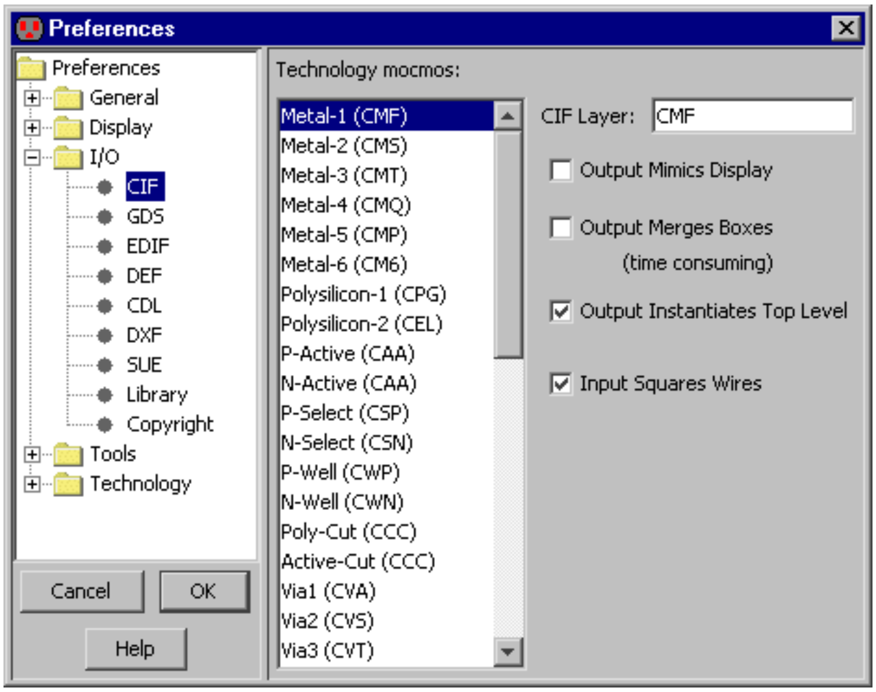
Unfortunately, many of these formats are pure geometry with no information about the circuit connections. When read, they appear as pure−layer nodes. This means that transistors, contacts, and other multi−layer nodes are not constructed properly. Although the cell appears visually correct, and can be used to export the same type of file, it cannot be analyzed at a circuit level. The node extractor can be used to convert these pure−layer nodes to true Electric components (see Section 9−10−2).

The next few sections describe control of different I/O formats.

7−3−2: CIF Control

CIF (Caltech Intermediate Format) is used as an interchange between design systems and fabrication facilities. For information on reading and writing CIF, see Section 3−9−2 and Section 3−9−3, respectively. CIF options are controlled with the "CIF" preferences (in menu **File / Preferences...**, "I/O" section, "CIF" tab).

This dialog controls the conversion between layers in Electric and layers in the CIF file. By clicking on an Electric layer, you can type a new CIF layer name into the dialog.



Technology mocmos:

Metal-1 (CMF)
Metal-2 (CMS)
Metal-3 (CMT)
Metal-4 (CMQ)
Metal-5 (CMP)
Metal-6 (CM6)
Polysilicon-1 (CPG)
Polysilicon-2 (CEL)
P-Active (CAA)
N-Active (CAA)
P-Select (CSP)
N-Select (CSN)
P-Well (CWP)
N-Well (CWN)
Poly-Cut (CCC)
Active-Cut (CCC)
Via1 (CVA)
Via2 (CVS)
Via3 (CVT)

CIF Layer: CMF

☐ Output Mimics Display

☐ Output Merges Boxes (time consuming)

☑ Output Instantiates Top Level

☑ Input Squares Wires

By default, CIF output writes the entire hierarchy below the current cell. If you check the "Output Mimics Display" item, cell instances that are unexpanded will be represented as an outline in the CIF file. This is useful when the CIF output is intended for hardcopy display, and only the screen contents is desired.

Another option is whether or not to merge adjoining geometry. This is an issue because of the duplication and overlap that occurs wherever arcs and nodes meet. The default action is to write each node and arc individually. This makes the file larger because of redundant box information, however it is faster to generate and uses simpler constructs. If you check the "Output Merges Boxes" item, all connecting regions on the same layer are merged into one complex polygon. This requires more processing, produces a smaller file, and generates more complex constructs.

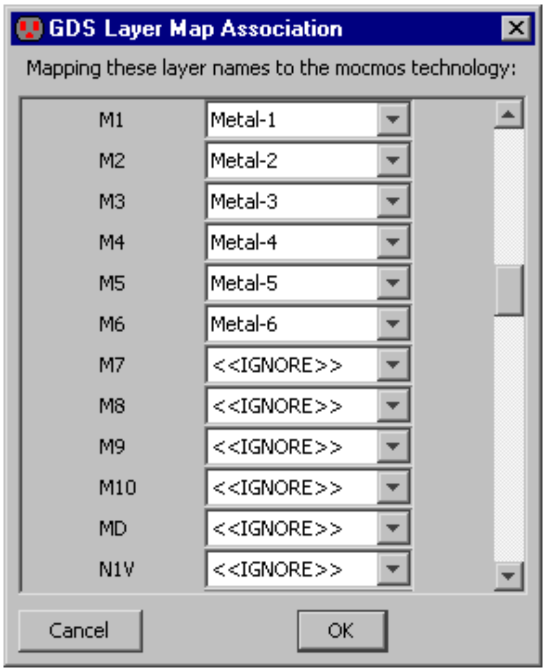
Another option is whether or not to instantiate the circuit in the CIF. By default, the currently displayed cell becomes the top level of the CIF file, and is instantiated at the end of the CIF. This causes the CIF file to display the current cell. If the CIF file is to be used as a library, with no current cell, then uncheck the "Output Instantiates Top Level" checkbox, and there will be no invocation of the current cell.

When reading CIF files, the CIF "wire" statements are assumed to have rounded geometry at the ends and corners. If you check the "Input Squares Wires" item, CIF input assumes that wire ends are square and extend by half of their width.

Be advised that the CIF format has a minimum resolution of 10 nanometers. Since nothing smaller can be accurately represented in the file, the CIF output of smaller geometries will generate errors.

7−3−3: GDS Control

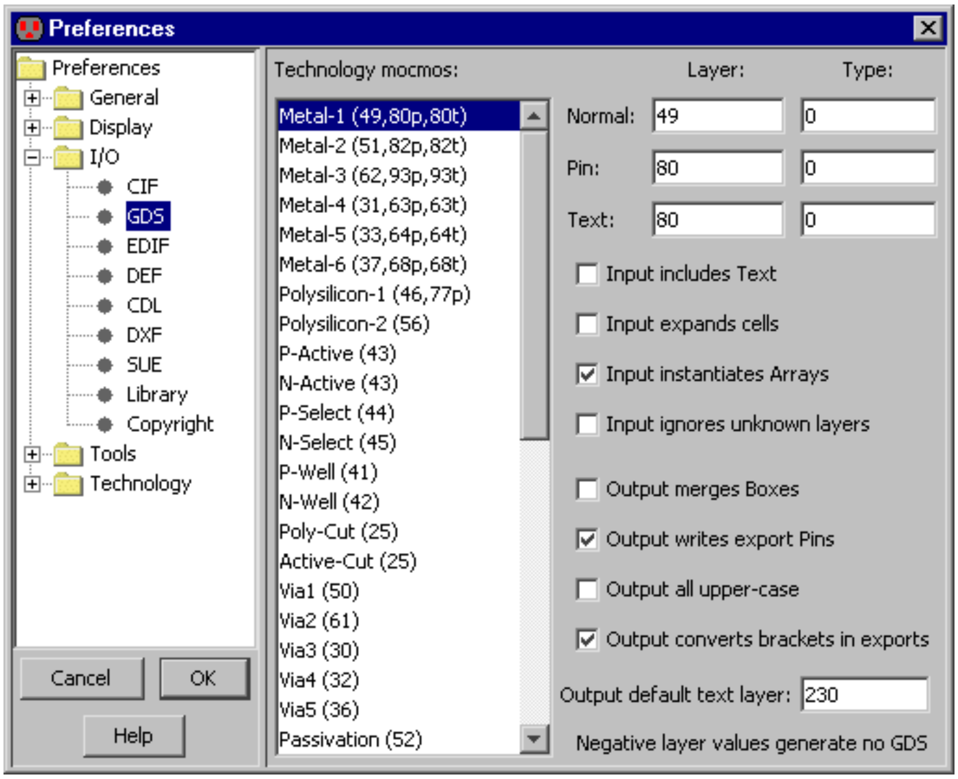
GDS II (also called "Stream" format) is used as an interchange between design systems and fabrication facilities. For information on reading and writing GDS, see Section 3−9−2 and Section 3−9−3, respectively. In GDS files, there are no names for each layer, just a pair of numbers (the layer number and type). It is important that Electric know how these values correspond with layers so that it can properly read and write GDS files. You can set the correspondences by using the **GDS Map File...** command (in menu **File / Import**) to read a GDS map file. You can also use the "GDS" preferences dialog (in menu **File / Preferences...**, "I/O" section, "GDS" tab) to edit the GDS numbers and control other aspects of GDS input and output.



Mapping these layer names to the mocmos technology:

M1	Metal-1
M2	Metal-2
M3	Metal-3
M4	Metal-4
M5	Metal-5
M6	Metal-6
M7	<<IGNORE>>
M8	<<IGNORE>>
M9	<<IGNORE>>
M10	<<IGNORE>>
MD	<<IGNORE>>
N1V	<<IGNORE>>

In the "GDS" preferences dialog, the list on the left shows all of the Electric layers in the current technology. By clicking on a layer name, its GDS numbers are shown in the top−right and can be edited. In addition to GDS numbers to use for layout, there are also two other types of GDS numbers: *pin* (for exports) and *text* (for export names).


Preferences

- Preferences
  - General
  - Display
  - I/O
    - CIF
    - GDS
    - EDIF
    - DEF
    - CDL
    - DXF
    - SUE
    - Library
    - Copyright
  - Tools
  - Technology

Cancel | OK | Help

Technology mocmos:

Metal-1 (49,80p,80t)
Metal-2 (51,82p,82t)
Metal-3 (62,93p,93t)
Metal-4 (31,63p,63t)
Metal-5 (33,64p,64t)
Metal-6 (37,68p,68t)
Polysilicon-1 (46,77p)
Polysilicon-2 (56)
P-Active (43)
N-Active (43)
P-Select (44)
N-Select (45)
P-Well (41)
N-Well (42)
Poly-Cut (25)
Active-Cut (25)
Via1 (50)
Via2 (61)
Via3 (30)
Via4 (32)
Via5 (36)
Passivation (52)

	Layer:	Type:
Normal:	49	0
Pin:	80	0
Text:	80	0

☐ Input includes Text
☐ Input expands cells
✓ Input instantiates Arrays
☐ Input ignores unknown layers

☐ Output merges Boxes
✓ Output writes export Pins
☐ Output all upper-case
✓ Output converts brackets in exports

Output default text layer: 230

Negative layer values generate no GDS

These dialog elements apply to reading GDS:

- "Input includes Text". Text annotations in the GDS file can often clutter the display, so they are ignored during input. If you check this item, annotation text will be read and displayed.
- "Input expands cells". This controls whether cell instances are expanded or not in the Electric circuit. By default, cell instances are not expanded (they appear as a simple box). If you check this item, cells are expanded so that their contents are displayed. Expansion of cells can always be changed after reading GDS by using the subcommands of the **Expand Cell Instances** and **Unexpand Cell Instances** commands of the **Cells** menu.
- "Input instantiates Arrays". This controls whether or not arrays in the GDS file are instantiated. By default, arrays are instantiated fully, but this can consume excessive amounts of memory if there are large arrays. If you uncheck this item, only the upper−left and lower−right instance are actually placed.
- "Input ignores unknown layers". This controls whether unknown layers in the GDS file will be ignored, or placed in the circuit. By default, unknown layers appear as DRC−Nodes (special nodes used to indicate DRC errors, which appear as orange squares). By checking this item, the unknown layers are simply ignored.

These dialog elements apply to writing GDS:

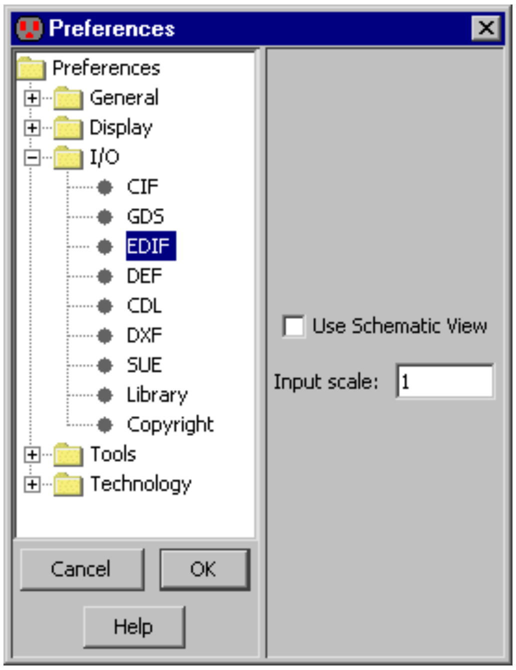
- "Output merges Boxes". This controls the merging of adjoining geometry. This is an issue because of the duplication and overlap that occurs wherever arcs and nodes meet. The default action is to write each node and arc individually. This makes the file larger because of redundant box information, however it is faster to generate and uses simpler constructs. If you check this item, all connecting regions on the same layer are merged into one complex polygon. This requires more processing, produces a smaller file, and generates more complex constructs.
- "Output Writes export Pins". This controls whether pins are written to the GDS file for each export. If checked, and there is a valid pin layer, then it is written.

- "Output all upper case". This controls whether the GDS file uses all upper case. The default is to mix upper and lower case, but some systems insist on upper−case GDS.
- "Output converts brackets in exports". This controls whether the square brackets used in array specifications should be converted (to underscores). Some GDS readers cannot handle the square bracket characters.
- "Output default text layer". This is the layer number to use when writing text. When exports are being written, and there is a text layer number associated with the appropriate Electric layer, then that layer number is used instead of this default number.

7−3−4: EDIF Control

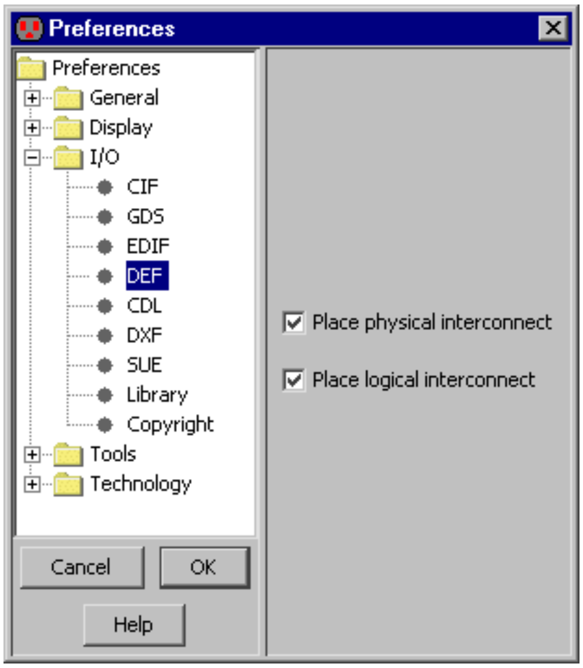
EDIF (Electronic Design Interchange Format) is used to exchange design information between different CAD systems. Although EDIF is currently at version "4 0 0", Electric reads and writes version "2 0 0". For more information on reading and writing EDIF, see Section 3−9−2 and Section 3−9−3, respectively. EDIF options are controlled with the "EDIF" preferences (in menu **File / Preferences...**, "I/O" section, "EDIF" tab).

This dialog controls whether EDIF output writes schematic or netlist views (the default is netlist). It also lets you set a scale factor for EDIF input.



☐ Use Schematic View
Input scale: 1

7−3−5: DEF Control



[✓] Place physical interconnect

[✓] Place logical interconnect

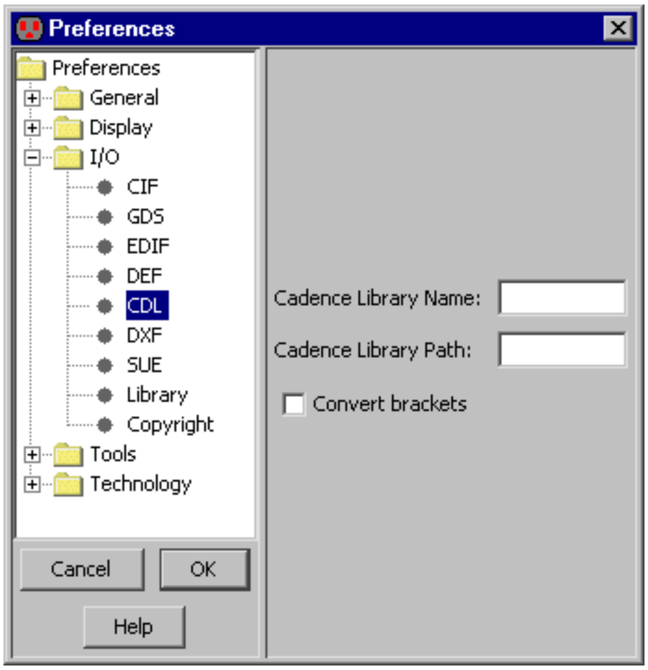
DEF (Design Exchange Format) is a recent interchange format for CAD systems. It is often combined with LEF (Library Exchange Format) files. For more information on reading and writing DEF or LEF, see Section 3−9−2 and Section 3−9−3, respectively. DEF options are controlled with the "DEF" preferences (in menu **File / Preferences...**, "I/O" section, "DEF" tab).

This dialog controls whether DEF reads physical and/or logical information.

7−3−6: CDL Control

CDL (Circuit Description Language) is almost identical to Spice format, and is used as a netlist interchange method. CDL options are controlled with the "CDL" preferences (in menu **File / Preferences...**, "I/O" section, "CDL" tab).

This dialog control the library name and path information that is written, and it lets you control the conversion of square−bracket characters.



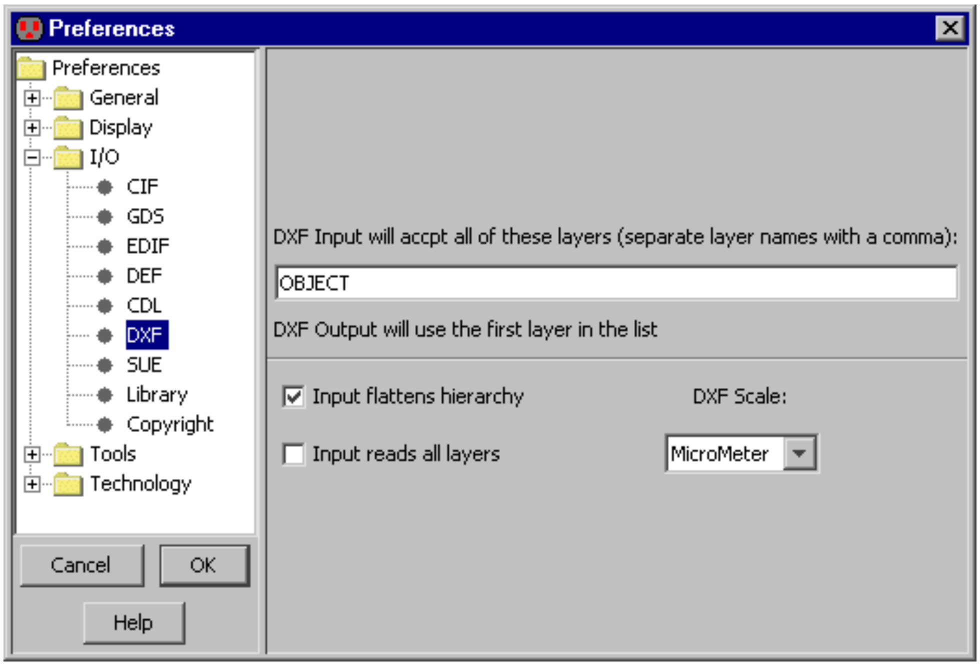
Cadence Library Name:

Cadence Library Path:

[ ] Convert brackets

7−3−7: DXF Control

DXF (Drawing eXchange Format) is a solid modeling format used by AutoCAD systems. For more information on reading and writing DXF, see Section 3−9−2 and Section 3−9−3, respectively. DXF options are controlled with the "DXF" preferences (in menu **File / Preferences...**, "I/O" section, "DXF" tab).



DXF Input will accpt all of these layers (separate layer names with a comma):

OBJECT

DXF Output will use the first layer in the list

[✓] Input flattens hierarchy

[ ] Input reads all layers

DXF Scale: MicroMeter

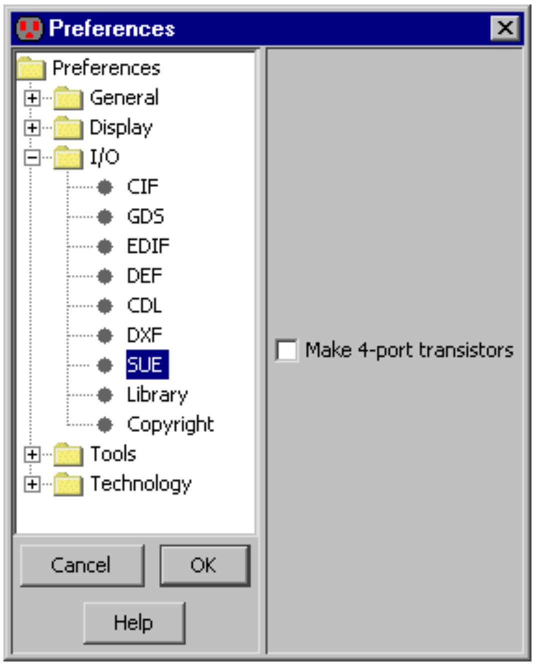
This dialog controls the list of acceptable DXF layers. These layers can be typed into the edit field, separated by commas. If a layer name in the DXF file is not found in the list of acceptable layers, it will be ignored. If you check "Input reads all layers", then all layers are read into Electric, regardless of whether the layer names are known.

By default, Electric flattens DXF input, removing levels of hierarchy and creating a single cell with the DXF artwork. By unchecking the "Input flattens hierarchy", Electric will preserve the structure of the DXF file.

To control scaling, you can change the meaning of units in the DXF file. The default unit is "Millimeters", which means that a value of 5 in the DXF file becomes 5 millimeters in Electric.

7−3−8: SUE Control

SUE (Schematic User Environment) is the database format of the SUE schematic editor, from Micro Magic. For more information on reading SUE, see Section 3−9−2. SUE options are controlled with the "SUE" preferences (in menu **File / Preferences...**, "I/O" section, "SUE" tab).



☐ Make 4-port transistors

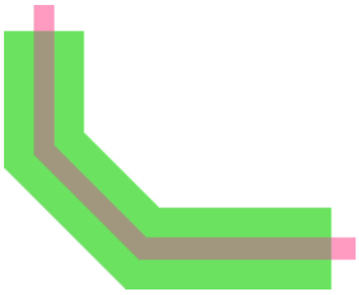
This dialog controls whether transistors will appears in a standard 3−terminal configuration or in a 4−port configuration with a substrate connection.

7−4−1: The MOS Technologies

There are both nMOS and CMOS technologies available in Electric, with many different design rules. Use the popup at the top of the component menu to select a different MOS technology.

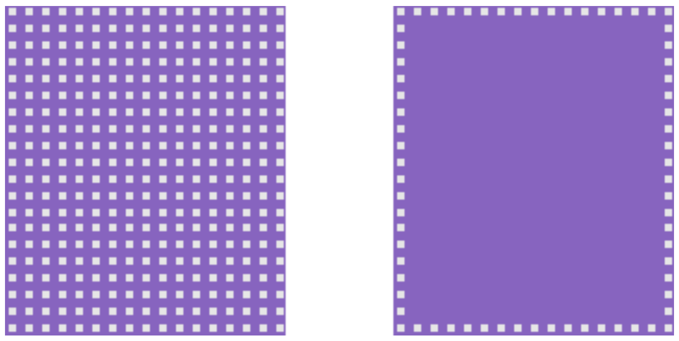
There is one nMOS technology: "nmos" (the specifications used in the Mead and Conway textbook).

There are a few CMOS technologies available. The most basic is "cmos", which uses an idealized set of design−rules from a paper by Griswold. The most popular CMOS technology is "mocmos" (MOSIS design rules) which has two layers of polysilicon and up to 6 layers of metal with standard, submicron, or deep rules (this is described more fully in the next section). There is even "rcmos", which uses round geometry!



Each MOS technology has two transistors (enhancement and depletion in nMOS technologies, *n* and *p* in CMOS). These nodes can have serpentine paths by highlighting them and using "Outline Edit" mode (see Section 6−10−1).

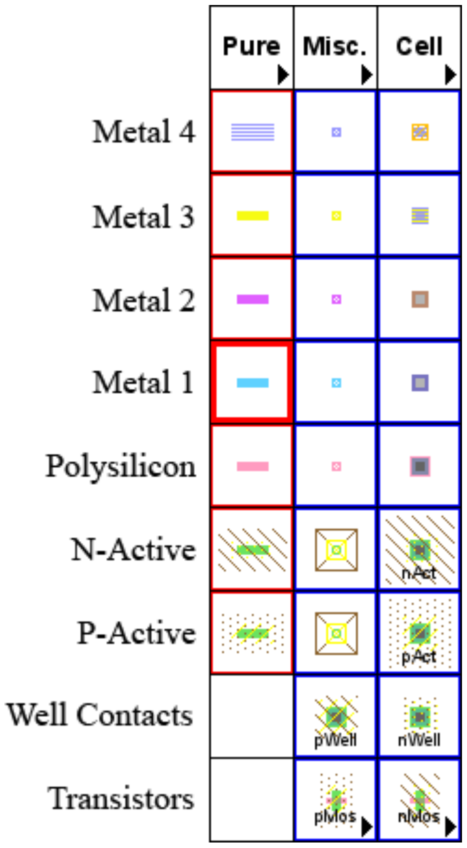
The contact cuts in the MOS technologies automatically increase the number of cut layers when the contact grows in size. For very large contacts, however, the display of these cuts can waste time. Therefore, when very large contacts are displayed at small scale, the interior cuts may not be drawn (as shown on the right). Be assured, however, that the cuts are actually there, and will appear in all appropriate output.



Although individual MOS nodes and arcs have the proper amount of implant around them, a collection of such objects may result in an irregular implant boundary. To clean this up, you can place pure−layer nodes of implant that neatly cover the implant area. Also, you can do this automatically with the **Coverage Implants Generator** command (in menu **Tool / Generation**).

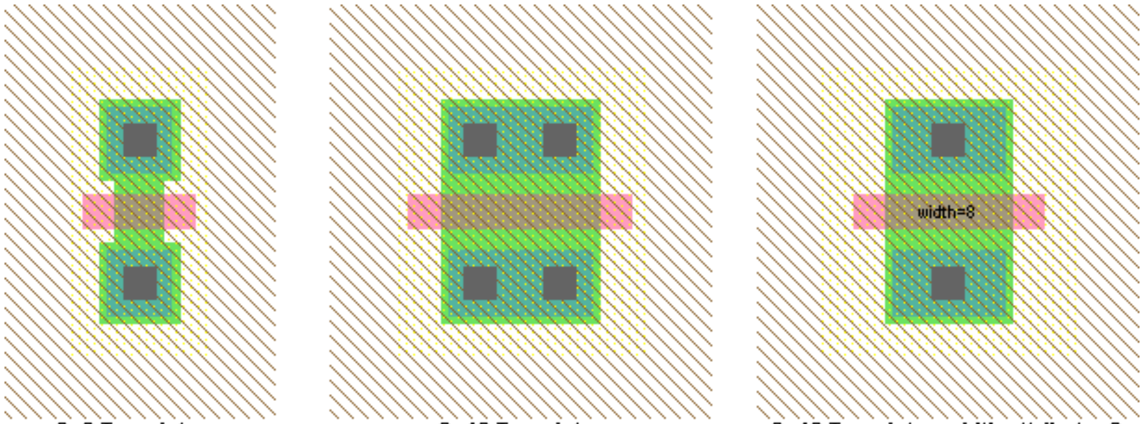
7−4−2: The MOSIS CMOS Technology

The MOSIS CMOS technology describes a scalable CMOS process that is fabricated by the MOSIS project of the University of Southern California. To obtain this technology, use the popup menu at the top of the component tab (in the side bar) and select "mocmos".



This technology defaults to 4 metal layers (shown here), but can also be changed so that it uses anywhere from 2 to 6 layers of metal. It also has 1 polysilicon layer but can be changed to use 2. The technology can also be set to use either standard rules (SCMOS), submicron rules, or deep rules. You can choose whether to allow stacked vias and whether or not to use alternate contact rules. All of this is done with the "Technology" preferences (in menu **File / Preferences...**, "Technology" section, "Technology" tab).

The MOSIS CMOS technology also has two scalable transistor nodes that can be parameterized to have different widths. The scalable transistors have contacts built into them. When created and scaled, their maximum width is shown. However, by adding a "width" attribute, they can shrink arbitrarily. Note that the ports remain in the same location regardless of the width, thus allowing them to scale without affecting constraints.

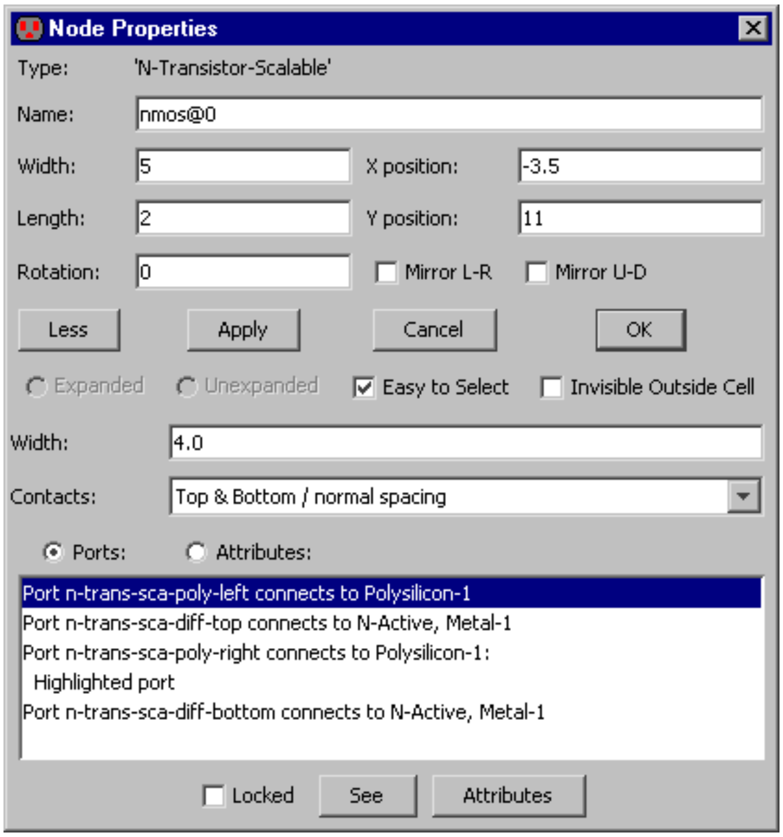


2x3 Transistor 2x10 Transistor 2x10 Transistor, width attribute=8

The scalable transistor on the left is 3 wide, and the other two are 10 wide. However, the scalable transistor on the right has had the "width" attribute set to 8 and so it has shrunk. Note that this attribute can be derived from cell parameters, causing different instances of the same cell to have different size transistors in it.

If you get **Object Properties...** on a scalable transistor, there are extra controls that let you choose to have fewer contacts (1 or even none), and you can tighten the contact spacing.

Users of Electric version 6.02 or earlier will have a different MOSIS CMOS technology called "mocmossub". This technology attempted to match the submicron rule set, but did not do so as accurately as the current "mocmos" technology. If you have designs in that technology, they will be automatically converted to the new "mocmos" when read in.

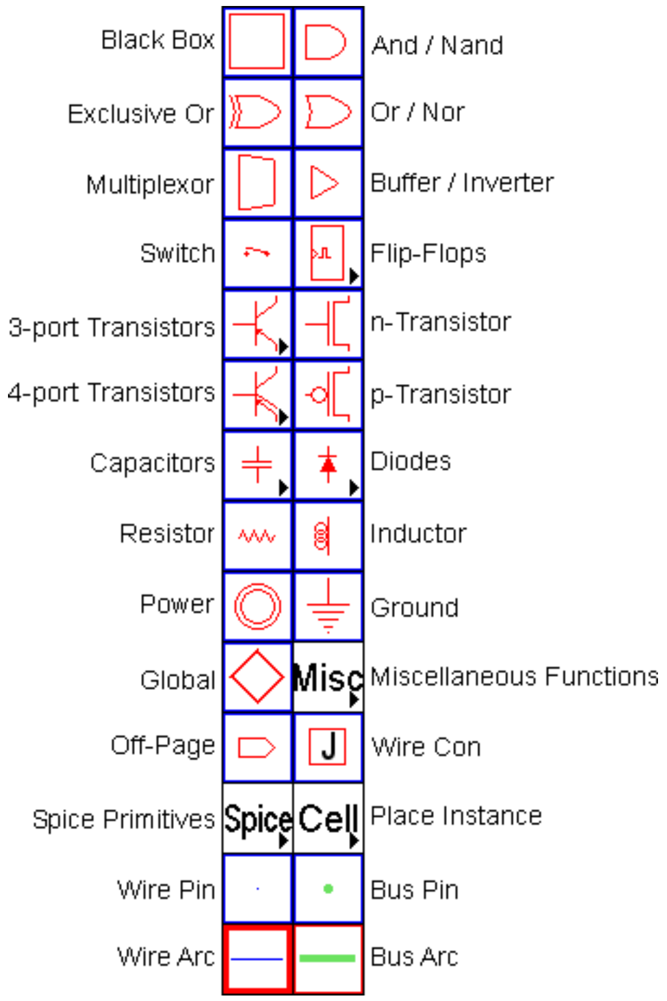


Type: 'N-Transistor-Scalable'
Name: nmos@0
Width: 5 X position: -3.5
Length: 2 Y position: 11
Rotation: 0 ☐ Mirror L-R ☐ Mirror U-D
○ Expanded ○ Unexpanded ☑ Easy to Select ☐ Invisible Outside Cell
Width: 4.0
Contacts: Top & Bottom / normal spacing
● Ports: ○ Attributes:
Port n-trans-sca-poly-left connects to Polysilicon-1
Port n-trans-sca-diff-top connects to N-Active, Metal-1
Port n-trans-sca-poly-right connects to Polysilicon-1:
Highlighted port
Port n-trans-sca-diff-bottom connects to N-Active, Metal-1
☐ Locked

7−5−1: The Schematics Technology

The Schematic technology allows you to design using digital and analog schematic components. To obtain this technology, use the popup menu at the top of the component menu and select "schematics".

There are two arcs in the Schematic technology: the wire (blue) and the bus (green). These arcs can be drawn at 45 degree angles. One typically names busses with array names (for example "insig[0:7]"), and then names wires with scalar names (for example "insig[1]"). See Section 6−9−3 for more on bus naming.



To make a physical connection of a wire to a bus, the bus pin can connect to either, so it acts as a tap. In addition, the Wire Con node connects wires to busses, or connects busses of different width, replicating the narrower side to make it as wide as the wider side. Use the **Rip Bus** command (in menu **Edit / Arc**) to automatically add taps to a bus.

Digital schematics are built with the And, Or, Xor, Buffer, Multiplexor, and Flip−Flop nodes that appear in the component menu. By attaching arcs to these components and negating them (with the **Toggle Port Negation** command, in menu **Edit / Technology Specific**), these turn into NAND, NOR, Inverter, and many other specialized components. Note that the size of the negating bubble can be controlled by using the "Technology" preferences (in menu **File / Preferences...**, "Technology" section, "Technology" tab), and setting the "Negating Bubble Size" value in the "Schematics" area.

The And, Or, Xor, and Multiplexor nodes can accept any number of input connections on the left, so they require some care in wiring (see Section 1−11−5). The left side has one large input port that allows an arbitrary number of connections. Initially, wires may attach at only three input locations, spaced evenly along the left side. However, when all three locations are connected, the node automatically expands, adding additional space along the side for new arcs.

To properly wire inputs to an And, Or, Xor, or Multiplexor node, cursor placement is very important, for it determines which of the locations to use on the left side. If an arc gets connected in the wrong location, try connecting more arcs until one appears in the right place, and then delete the unwanted ones.

The Switch node can also take an arbitrary number of poles on its left side. Simply stretch it along the line of the poles and their number will grow.

The analog nodes (Resistor, Inductor, Capacitor, and Diode) have values on them which can be selected and edited. Double−clicking on them brings up a special dialog for editing their value.

There are four transistor entries in the menu. The two on the right are the n and p transistors. The two images on the left are actually popup menus that let you select any style of transistor. The difference between the two on the left is that the top one is for 3−port transistors, and the bottom one is for 4−port transistors.

The "Spice" entry presents a popup menu of Spice parts . More information about the use of these parts can be found in the Section 9−4−3.

The "Cell" entry presents a popup menu of all cell instances.

The "Global" entry provides two nodes: a "Global Signal" node defines a signal name that spans levels of hierarchy , and a "Global Partition" node allows globals to be treated locally. See Section 6−9−5 for more on global networks.

The Resistor can be treated as a connecting or nonconnecting node. By default, it does not connect the networks on its two ends, and this is the correct way to treat it when doing low−level simulation such as Spice. However, for higher−level simulations (such as Verilog) the resistor should be ignored and treated as if it connects its two networks. To make this happen, use the "Network" preferences (in menu **File / Preferences...**, "Tools" section, "Network" tab), and check "Ignore Resistors". Note that if resistors are being ignored, Spice deck generation will temporarily include them while the netlist is being created.

Some commands that analyze a schematic circuit need to know which layout technology will be used to fabricate the design. For example, when generating a Spice deck from a schematic, it is necessary to know the sizes and parasitics that are associated with the actual circuit. To set the layout technology to use for schematic circuits, use the "Technology" preferences (in menu **File / Preferences...**, "Technology" section, "Technology" tab), and set the "Use scale values from this technology" popup.

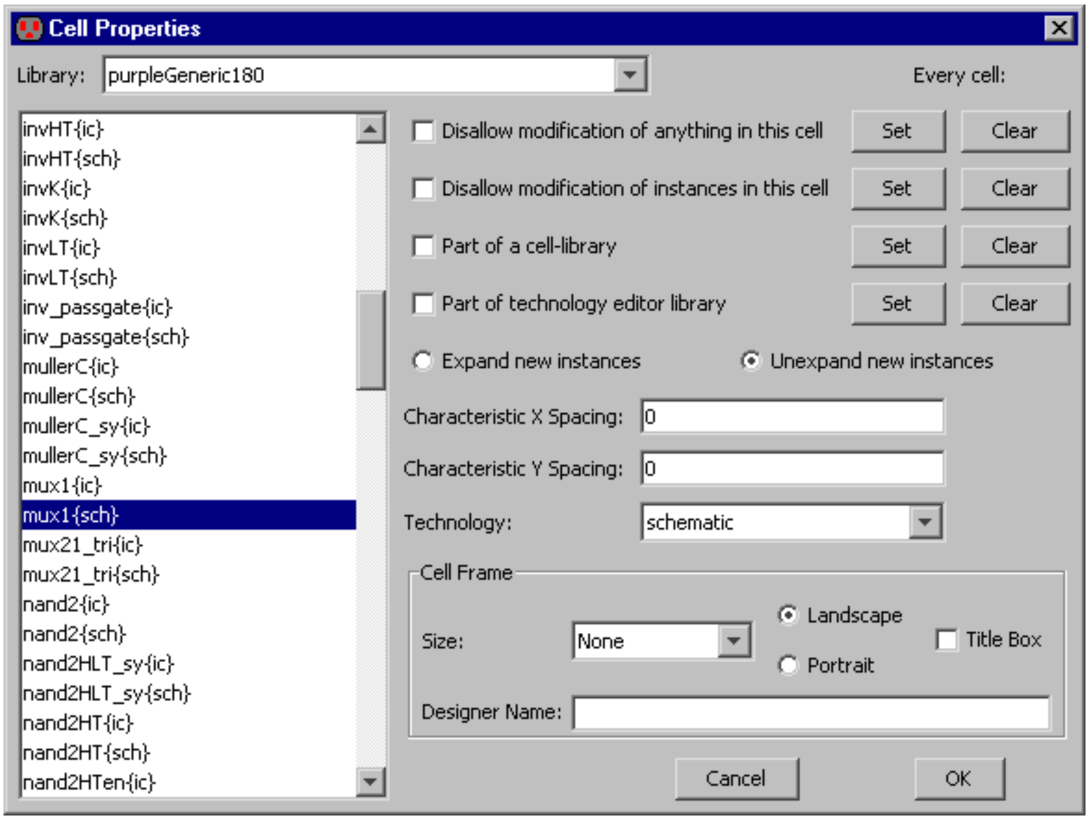
7−5−2: Multipage Schematics and Frames

Multipage schematics are implemented in Electric by having each page map to a different area of a vast schemetic cell. To create one of these multipage cells, use the **Make Cell Multi−Page** command (in menu **Cell / Multi−Page Cells**). You will then be editing page 1 of the multi−page schematic.

You can add pages to the current multipage schematic with the **Create New Page** command (in menu **Cell / Multi−Page Cells**). You can delete the current page with **Delete This Page**. To advance to the next page, use **Edit Next Page**.

Older versions of Electric implemented multipage schematics with different view types ("p1", "p2", ...). If these views appear instead of proper pages, use the **Convert old−style Multi−Page Schematics** command.

As a graphical aid to schematic design, frames can be displayed in a cell by using the **Cell Properties...** command (in menu **Cell**). Multi−page schematics require a cell frame on every page, but their presence is optional in other cells.



Cell Properties

Library: purpleGeneric180

Every cell:

invHT{ic}
invHT{sch}
invK{ic}
invK{sch}
invLT{ic}
invLT{sch}
inv_passgate{ic}
inv_passgate{sch}
mullerC{ic}
mullerC{sch}
mullerC_sy{ic}
mullerC_sy{sch}
mux1{ic}
mux1{sch}
mux21_tri{ic}
mux21_tri{sch}
nand2{ic}
nand2{sch}
nand2HLT_sy{ic}
nand2HLT_sy{sch}
nand2HT{ic}
nand2HT{sch}
nand2HTen{ic}

☐ Disallow modification of anything in this cell — Set / Clear

☐ Disallow modification of instances in this cell — Set / Clear

☐ Part of a cell-library — Set / Clear

☐ Part of technology editor library — Set / Clear

Expand new instances [✓ Unexpand new instances]

Characteristic X Spacing: 0

Characteristic Y Spacing: 0

Technology: schematic

Cell Frame

Size: None

[✓ Landscape] Portrait

☐ Title Box

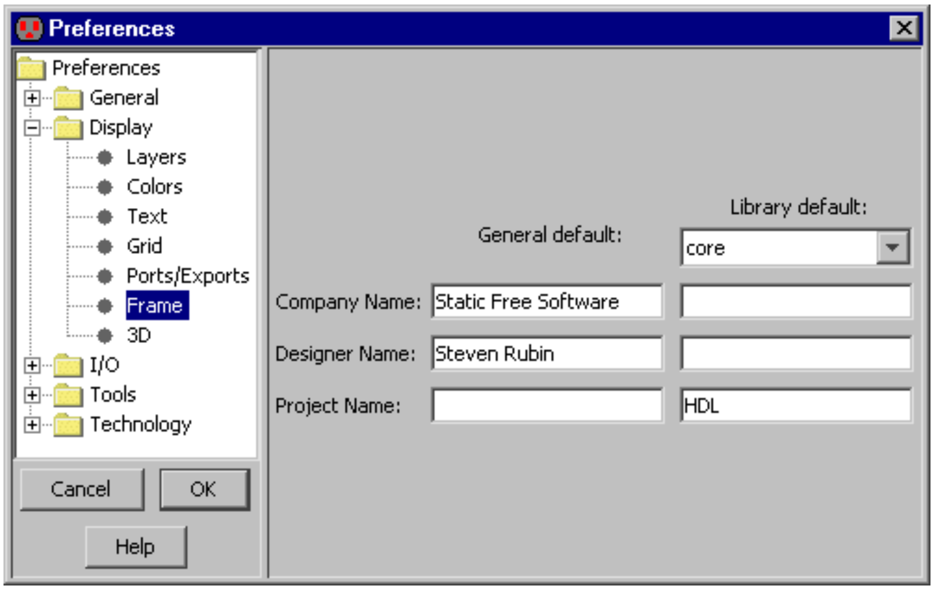
Designer Name:

Cancel OK

The frame size can be "Half−A", "A", "B", "C", "D", and "E". The frame can be horizontal (landscape) or vertical (portrait). You can choose to display a title box in the lower−right corner. The designer name can also be set for each cell.

Besides the designer name, cell frames have a company name and a project name. These values are not set for each cell, but instead are preferences that are set for each user. Individual libraries can override these defaults as well.

The "Frame" preferences (in menu **File / Preferences...**, "Display" section, "Frame" tab) lets you set all of these defaults. Note that the designer name is taken first from the cell, then from the library if the cell does not set a value, and finally from the general default if the library and cell do not set a value.



Preferences

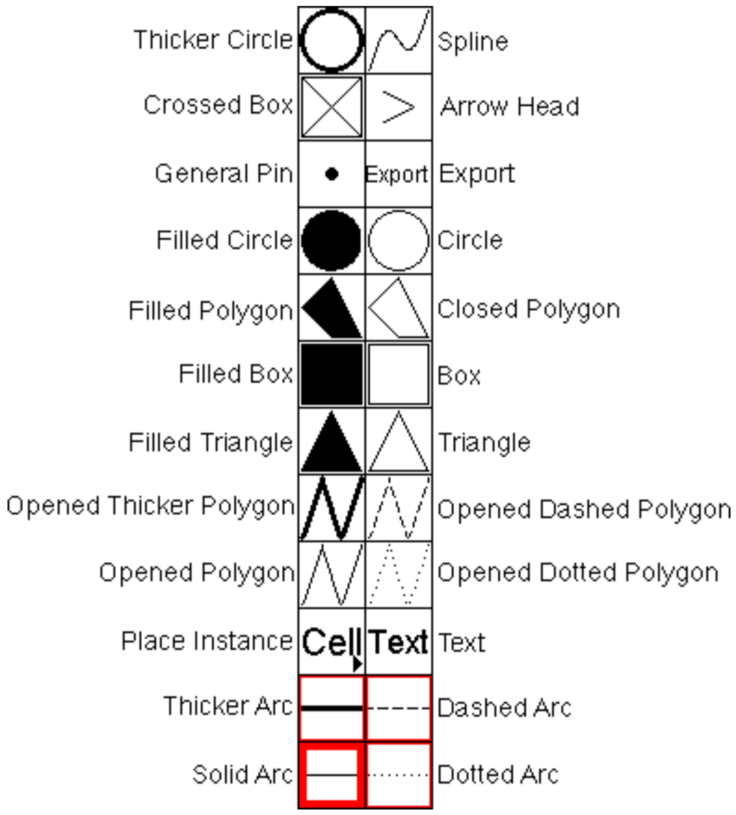
Preferences
General
Display
Layers
Colors
Text
Grid
Ports/Exports
Frame
3D
I/O
Tools
Technology

Cancel OK
Help

	General default:	Library default: core
Company Name:	Static Free Software	
Designer Name:	Steven Rubin	
Project Name:		HDL

7−6−1: The Artwork Technology

The Artwork technology is an unusual technology that provides general−purpose sketching facilities. To obtain this technology, use the popup menu at the top of the component menu and select "artwork".



This technology has nodes for many typical graphic objects such as rectangles, triangles, circles, and arrowheads. Polygonal and Spline nodes allow arbitrary shapes to be defined. Of course, nodes from all other technologies can be used as special electronic symbols when artwork is generated. Conversely, these artwork nodes can be used to embellish designs done in all other technologies.

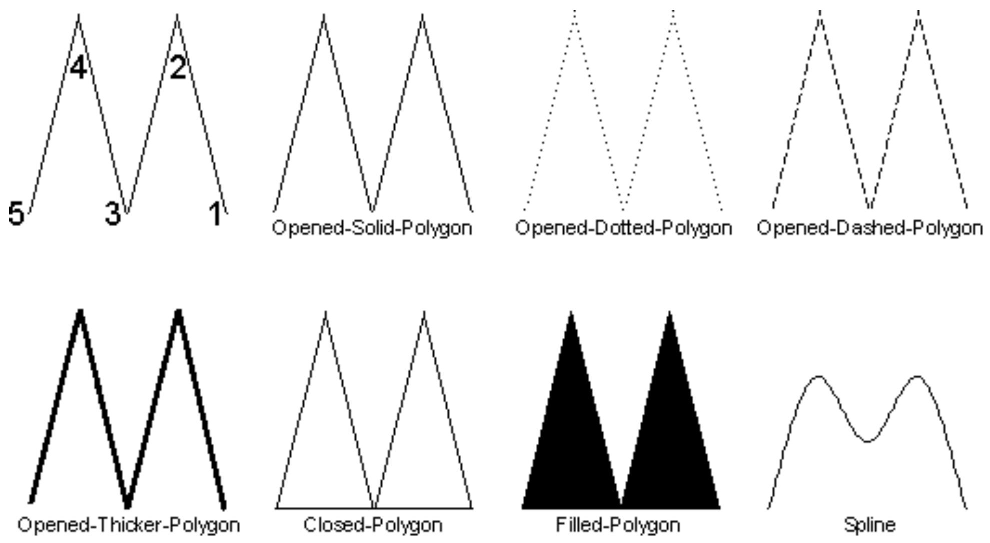
Circles can be outlines (normal or thick) or filled. The default shape is round, but elongation of the node produces an ellipse. In addition, by using the **Object Properties...** command (in menu **Edit / Properties**), the outline circles can be reduced to a portion of the circle (from 1 to 360 degrees).

Arrow heads can be drawn in two different styles: simple or filled. The simple arrow head is the default and consists of two lines. The filled arrow head looks better because it is made of polygons. Use the "Technology" preferences (in menu **File / Preferences...**, "Technology" section, "Technology" tab), and set the "Arrows filled" checkbox in the "Artwork" area.

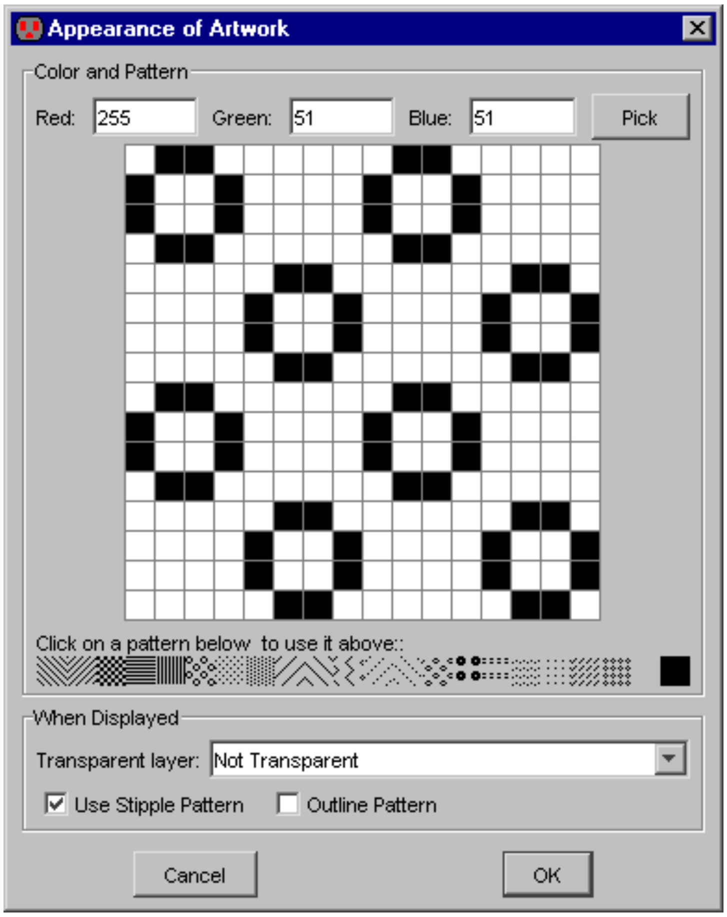
The "Export" entry creates an export for use in icons. After clicking on the entry, you have the choice of selecting "Wire", "Bus", or "Universal" exports (see Section 3−11−4 for more on icon generation).

There are four different polygon styles: opened, closed, filled, and spline. The opened polygon can be drawn with solid lines, dotted lines, dashed lines, or thicker lines. These nodes require that you use the "Outline Edit" mode (see Section 6−10−1).

The illustration below shows how outline information, applied to Artwork nodes, results in different shapes. In each of the shapes, the outline has the same 5 points, as illustrated in the upper−left. The nodes interpret this outline information to produce their shape. Note that the spline curve does not run through the outline points, only near them.



The final feature of the Artwork technology is its ability to set the appearance of any of its nodes or arcs. Use the **Artwork Appearance...** command (in menu **Edit / Technology Specific**) to set the color and pattern of any Artwork node or arc. Predefined patterns are available below the pattern−editing area. The transparent colors are taken from the current color map, which in turn is taken from the most recently selected technology (other than the Artwork technology).


Appearance of Artwork

Color and Pattern

Red: 255 Green: 51 Blue: 51

Click on a pattern below to use it above::

When Displayed

Transparent layer: Not Transparent

[✓] Use Stipple Pattern [ ] Outline Pattern

7−6−2: The FPGA Technology

The FPGA technology is a "soft" technology that creates primitives according to an FPGA Architecture file. Special commands in the **Edit / Technology Specific / FPGA** menu let you create the FPGA primitives, build FPGA structures, and program them.

The FPGA Architecture file contains all of the information needed to define a specific FPGA chip. It has three sections: the *Primitive Definition* section, the *Block Definition* section, and the *Arcitecture* section. The Primitive Definition section describes the basic blocks for a family of FPGA chips (these are primitives in the FPGA technology). The Block Definition section builds upon the primitives to create higher−level blocks. Finally, the Architecture section defines the top−level block that is the FPGA.

An FPGA Architecture file must have the Primitive Definition section, but it need not have the Block Definition or Architecture Sections. This is because the placement of the primitives can be saved in an Electric library, rather than the architecture file. Thus, after reading the Primitive Definition section (which creates the primitives), and reading the Block Definition and Architecture Sections (which places the primitives to create a chip library) the library can be saved to disk. Subsequent design activity can proceed by reading only the Primitive Definition section and then reading the library with the chip definition. This avoids large FPGA Architecture files (the Primitive Definition section will be smaller than the Block Definition and Architecture sections).

Primitive Definition Section

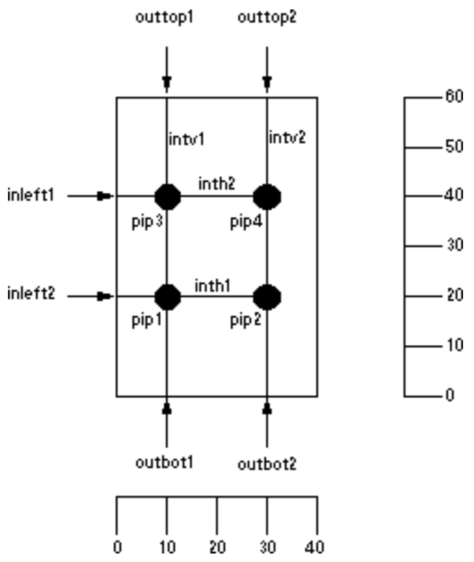
The Primitive Definition section defines the lowest−level blocks, which become primitive nodes in the FPGA technology. A primitive definition looks like this:

```
(primdef
  (attributes
    (name PRIMNAME)
    (size X Y)
  )
  (ports
    (port
      (name PORTNAME)
      (position X Y)
      (direction input | output | bidir)
    )
  )
  (components
    (pip
      (name PIPNAME)
      (position X Y)
      (connectivity NET1 NET2)
    )
  )
  (nets
    (net
      (name INTNAME)
      (segment FROMPART TOPART)
    )
  )
)
```

The **attributes** section defines general information about the block. The **ports** section defines external connections. The **components** section defines logic in the block (currently only PIPs). The **nets** section defines internal networks. There can be multiple **segment** entries in a net, each defining a straight wire that runs from the **FROMPART** to the **TOPART**. These parts can be either **port PORTNAME** or **coord X Y**, depending on whether the net ends at a port or at an arbitrary position inside of the primitive.

For example, this block has two vertical nets and two horizontal nets. Four pips are placed at the intersections. Six ports are defined (two on the left, two on the top, and two on the bottom). The code is as follows:

```
(primdef
  (attributes
    (name sampleblock)
    (size 40 60)
  )
  (ports
    (port (name inleft1) (position  0 40)
(direction input) )
    (port (name inleft2) (position  0 20)
(direction input) )
    (port (name outtop1) (position 10
60) (direction output) )
    (port (name outtop2) (position 30
60) (direction output) )
    (port (name outbot1) (position 10  0)
(direction output) )
    (port (name outbot2) (position 30  0)
(direction output) )
  )

  (components
    (pip (name pip1) (position 10 20)
(connectivity intv1 inth1) )
    (pip (name pip2) (position 30 20)
(connectivity intv2 inth1) )
    (pip (name pip3) (position 10 40)
(connectivity intv1 inth2) )
    (pip (name pip4) (position 30 40)
(connectivity intv2 inth2) )
  )

  (nets
    (net (name intv1) (segment port
outbot1 port outtop1 ) )
    (net (name intv2) (segment port
outbot2 port outtop2 ) )
    (net (name inth1) (segment port
inleft2 coord 30 20 ) )
    (net (name inth2) (segment port
inleft1 coord 30 40 ) )
  )
)
```



Block Definition and Architecture Sections

The Block Definition and Architecture sections define higher−level blocks composed of primitives. They looks like this:

```
(blockdef
  (attributes
    (name CHIPNAME)
    (size X Y)
    (wirecolor COLOR)
    (repeatercolor COLOR)
  )
  (ports
    (port
      (name PORTNAME)
      (position X Y)
      (direction input | output | bidir)
    )
  )
  (components
    (instance
      (attributes ATTPAIRS)
      (type BLOCKTYPE)
      (name BLOCKNAME)
      (position X Y)
      (rotation ROT)
    )
    (repeater
      (name BLOCKNAME)
      (porta X Y)
      (portb X Y)
      (direction vertical | horizontal)
    )
  )
  (nets
    (net
      (name INTNAME)
      (segment FROMPART TOPART)
    )
  )
)
```

The only difference between the Architecture section and the Block Definition section is that the Architecture section has the keyword **architecture** instead of **blockdef**. There can be only one **architecture** section, but there can be many **blockdef**s, defining a complete hierarchy.

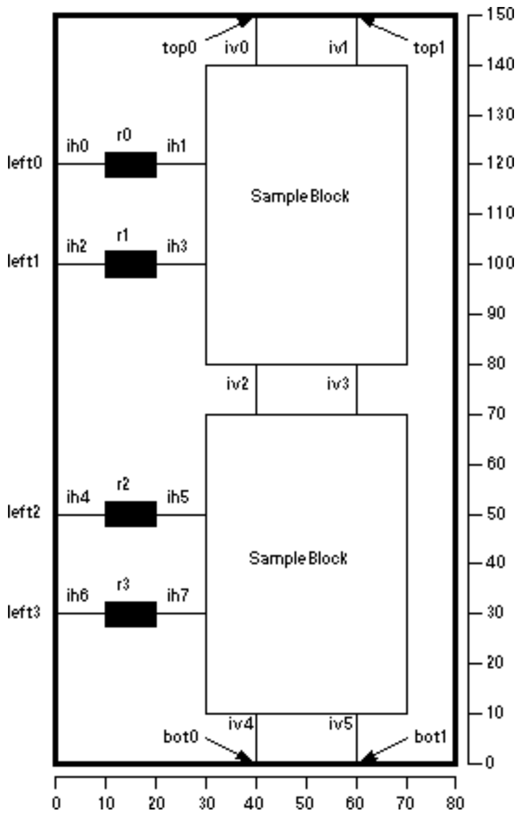
The **attributes** section defines general information about the block.

The **ports** section defines external connections.

The **components** section defines logic in the block (currently instances of other blocks or repeaters). The **rotation** of an instance is the number of degrees counterclockwise, rotated about the center. The **attributes** section of the instance assigns name/value pairs (this can be used to program the FPGA).

The **nets** section defines internal networks. There can be multiple **segment** entries in a net, each defining a straight wire that runs from the **FROMPART** to the **TOPART**. These parts can be either **component INSTNAME PORTNAME**, **port PORTNAME**, or **coord X Y**, depending on whether the net ends at a component, port or at an arbitrary position inside of the block.

Here is an example of block definition
code and its layout.



```
(blockdef
  (attributes
    (name testblock)
    (size 80 150)
  )
  (components
    (instance (type sampleblock) (name
block0)
      (position 30 80) )
    (instance (type sampleblock) (name
block1)
      (position 30 10) )
    (repeater (name r0) (porta 10 120)
      (portb 20 120) (direction
horizontal)
    )
    (repeater (name r1) (porta 10 100)
      (portb 20 100) (direction
horizontal)
    )
    (repeater (name r2) (porta 10 50)
      (portb 20 50) (direction
horizontal)
    )
    (repeater (name r3) (porta 10 30)
      (portb 20 30) (direction
horizontal)
    )
  )

  (ports
    (port (name top0)  (position 40 150) (direction bidir) )
    (port (name top1)  (position 60 150) (direction bidir) )
    (port (name left0) (position  0 120) (direction input) )
    (port (name left1) (position  0 100) (direction input) )
    (port (name left2) (position  0  50) (direction input) )
    (port (name left3) (position  0  30) (direction input) )
    (port (name bot0)  (position 40  0) (direction bidir) )
    (port (name bot1)  (position 60  0) (direction bidir) )
  )

  (nets
    (net (name iv0)
      (segment port top0 component block0 outtop1) )
    (net (name iv1)
      (segment port top1 component block0 outtop2) )
    (net (name iv2)
      (segment component block0 outbot1 component block1 outtop1))
    (net (name iv3)
      (segment component block0 outbot2 component block1 outtop2))
    (net (name iv4)
      (segment component block1 outbot1 port bot0) )
    (net (name iv5)
      (segment component block1 outbot2 port bot1) )
    (net (name ih0)
      (segment port left0 component r0 a) )
    (net (name ih1)
      (segment component r0 b component block0 inleft1) )
    (net (name ih2)
      (segment port left1 component r1 a) )
    (net (name ih3)
      (segment component r1 b component block0 inleft2) )
    (net (name ih4)
      (segment port left2 component r2 a) )
    (net (name ih5)
      (segment component r2 b component block1 inleft1) )
    (net (name ih6)
      (segment port left3 component r3 a) )
    (net (name ih7)
      (segment component r3 b component block1 inleft2) )
  )
)
```

Commands

To read an architecture file, use the **Read Architecture And Primitives...** command (in menu **Edit / Technology Specific / FPGA**). You will be prompted for an architecture file. To read only the primitives from an architecture file, use the **Read Primitives...** command.

Once an FPGA is on the screen, two aspects of its display can be controlled: the wires and the text. Three commands control the display of wires: **Show All Wires** displays every wire, **Show No Wires** hides every wire, and **Show Active Wires** shows only the wires that have been connected to PIPs that have been programmed. Two commands control the display of text: **Show Text** displays text and **Hide Text** turns text display off.

Once an FPGA has been created, you can program the PIPs by selecting a component and using the **Edit Pips...** command. This will display a list of active PIPs on the component. For example, after clicking on one of the "SampleBlock" instances, you can type the string "pip1 pip4" to program two of the pips in that instance.

7−6−3: The Generic Technology

One particularly interesting technology is the Generic technology, which is a grab bag of miscellaneous facilities. It is not necessary to actually switch into this technology, for all of its nodes and arcs are available through other means.

Special Arcs

The *Universal arc* in the Generic technology is able to make a connection between any two components, even if they are in different technologies. This is useful when mixing technologies while still maintaining proper connectivity, for example when simulating. The *Invisible arc* attaches any two components, but makes no electrical connection. It is useful for constraining otherwise unrelated components. The *Unrouted arc* makes arbitrary electrical connections, like the universal arc, but routers know to replace them with real geometry. None of these arcs produce any actual geometry in IC descriptions, but they make important conceptual connections.

Any existing arc in a normal technology can be converted to one of these three special arcs by using the **Change...** command (in menu **Edit**).

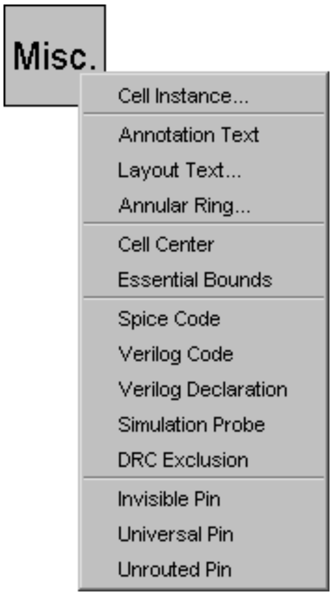
Special Nodes

There are also special nodes in the Generic technology. They are all available from the "Misc." entry of the component menu.

The *Universal−Pin* is a node that can connect to any arc. This is useful as an intermediate component when replacing (first you replace the unwanted node with a Universal−Pin to allow it to fit with the existing arcs; then you replace the arcs; finally you put the desired new node in place).

The *Invisible−Pin* is used for holding text, and it does not appear in hardcopy output (this is what is created when you use place Annotation Text). This pin can also connect to any arc.

A special primitive, called *Cell−Center*, defines the origin of any cell. Once the node is placed, its location is at (0,0) for the cell. Since instances of the current cell use the origin as the *anchor point* for cursor−based references, the location of this node defines the anchor. For example, if you place this node in the upper−right corner of a cell, then creation commands place instances such that their upper−right corner is at the cursor. See Section 3−3 for more information on cell centers.



A special primitive, called *Essential−Bounds*, defines an alternate boundary of any cell. At least two of them must be placed in opposite corners, although 4 can be place to make it look better.

Note that the Cell−Center and Essential−Bounds nodes are made "hard−to−select" by default, which means that they can be selected only by using "Special Select" mode (see Section 2−1−5 for more).

A special primitive, called *Simulation−Probe* is recognized by simulators and visually modified to reflect whatever it is connected to. The simulators that reflect the state of the circuit by drawing lines along arcs also fill−in these probe nodes. It provides a visual display of simulation activity, and works especially well with the VCR controls in the waveform window. See Section 4−12−1 for more.

Chapter 8: Creating New Technologies

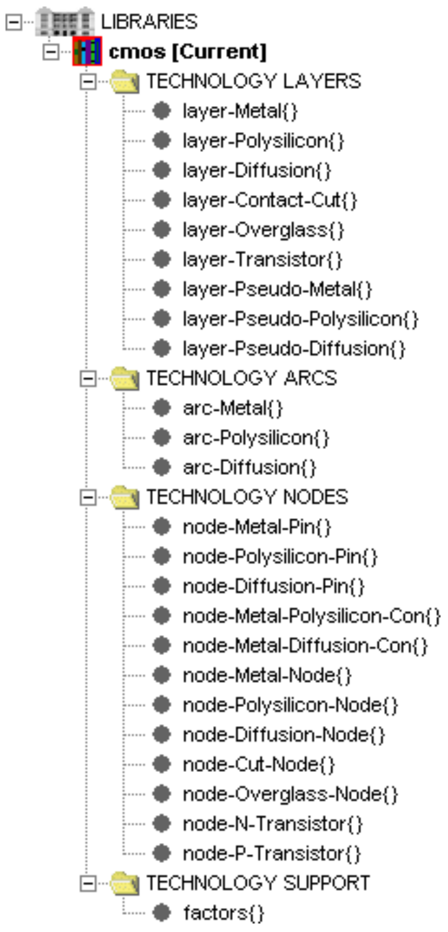
8−1: Designing New Technologies

Although there are many technology descriptions in Electric, there are many more in the world. To accommodate this, there is the *technology editor* which allows you to modify existing technologies and create new ones.

The editor works by converting a technology into a library of cells. You then edit the cells, using familiar Electric commands, and make changes to the technology. Finally, the technology editor translates the library back into a technology.

Libraries which describe a technology are called *technology libraries*. They use elements from the Artwork technology to describe their information. Special commands from the **Edit / Technology Specific** menu aid in the manipulation of these libraries.

There are four types of cells in a technology library which describe the *layers*, *arcs*, *nodes*, and *support*. They are separated into these groups in the cell explorer. The layer cells all begin with the name "layer−" and each one defines a layer in the technology. For example, the cell called "layer−Metal" defines the metal layer. The node and arc cells correspond to the primitives in the technology. Their names always begin with "node−" and "arc−". The support cell is always called "factors". Any other cell in the library is ignored.



```
LIBRARIES
  cmos [Current]
    TECHNOLOGY LAYERS
      layer-Metal{}
      layer-Polysilicon{}
      layer-Diffusion{}
      layer-Contact-Cut{}
      layer-Overglass{}
      layer-Transistor{}
      layer-Pseudo-Metal{}
      layer-Pseudo-Polysilicon{}
      layer-Pseudo-Diffusion{}
    TECHNOLOGY ARCS
      arc-Metal{}
      arc-Polysilicon{}
      arc-Diffusion{}
    TECHNOLOGY NODES
      node-Metal-Pin{}
      node-Polysilicon-Pin{}
      node-Diffusion-Pin{}
      node-Metal-Polysilicon-Con{}
      node-Metal-Diffusion-Con{}
      node-Metal-Node{}
      node-Polysilicon-Node{}
      node-Diffusion-Node{}
      node-Cut-Node{}
      node-Overglass-Node{}
      node-N-Transistor{}
      node-P-Transistor{}
    TECHNOLOGY SUPPORT
      factors{}
```

8−2: Converting between Technologies and Libraries

Converting Technologies to Libraries

The best way to create a new technology is to change an existing one. Use the **Convert Technology to Library for Editing...** command (in menu **Edit / Technology Specific**) and select a similar technology. Unfortunately, the Schematic and Artwork technologies are too complex to edit and cannot be converted.

Conversion of a technology to a library creates a library with the same name as the technology. Note that technologies with options (such as MOSIS CMOS) will be converted with their current settings only, and the options will no longer be available.

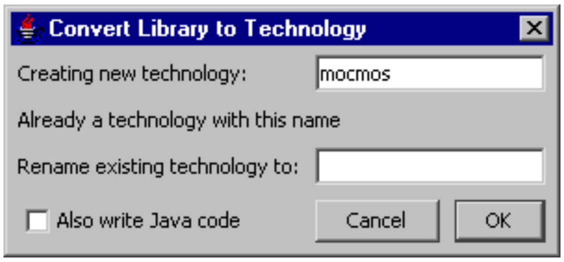
Technology−Editing Mode

Once a technology−library has been created, editing of its cells is done in a special *technology−editing mode*. The system knows to use technology−editing mode because the cells are marked as being "Part of a technology editor library" (see the **Cell Properties...** command of the **Cells** menu).

Converting Libraries to Technologies

To convert a technology−library into a technology, use the **Convert Library to Technology...** command.

You are given the opportunity of naming the technology, and can also request that Java code be produced (this code can be compiled with Electric to install the technology permanently). If a technology already exists with the name you want, you can request that it be renamed, or you can choose a different name for the new technology (note that technologies can also be renamed with the **Rename Current Technology...** command).


Convert Library to Technology
Creating new technology: mocmos
Already a technology with this name
Rename existing technology to:
☐ Also write Java code

If there is an error in the library, conversion is aborted and you are given a chance to fix the library. Generally, the offending part of the library is highlighted. If no errors have occurred in the translation, there will be a new technology in Electric and it will be the current one.

Before creating any circuitry with the new technology, it is advisable to create a new library (use the **New Library...** command of menu **File**) so that the test circuitry is not stored with the library that describes it.

Cleaning Up

After a few rounds of technology editing, there may be many libraries and technologies. You can delete the current library with the **Close Library** command of the **File** menu (to make another library current, use the **Change Current Library...** command of the **File** menu).

Using Technology Libraries

Once a library has been successfully built that describes a technology, it can be saved to disk with the **Save Library** command of the **File** menu. Then, in another session of Electric, it can be read from disk and converted to a technology.

8−3: Hierarchies of Technology Libraries

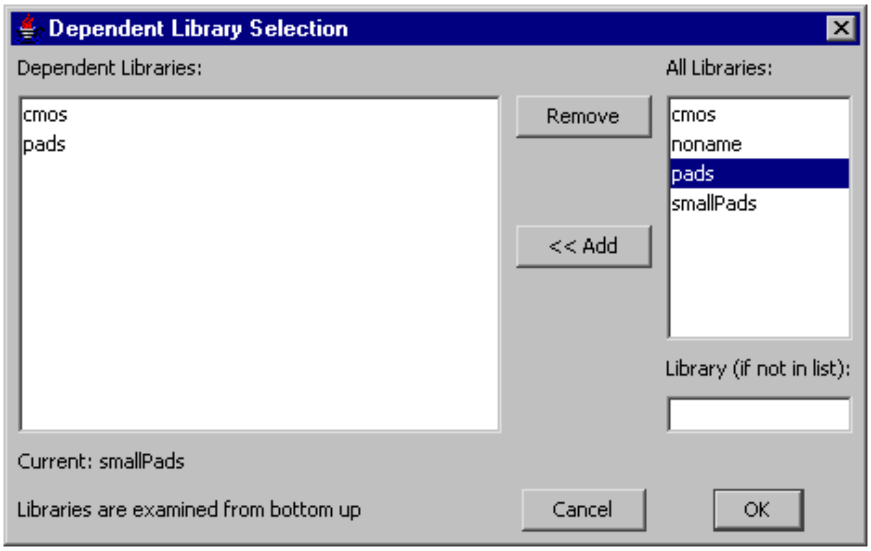
Although a technology is normally described with a single library, it is also possible to string together a sequence of libraries to describe a technology. The sequence forms an "inheritance hierarchy", where later libraries in the sequence can override elements found in earlier libraries. For example, one library could be a "base" description for a family of technologies, and another library could be a "tailoring" description that describes a specific family member. The tailoring library might be very small, consisting of a single node description. That information would then override or augment the base library.

To connect a sequence of libraries, a list is placed in the bottommost library pointing to the earlier, or "dependent" libraries. In the example below, the current library is "smallPads" and it is tailored with two other libraries: "pads" and "cmos" (the "base" library). Note that the list implicitly begins with the current library, and continues in reverse order. In this example, the first library examined is "padsSmall", followed by "pads" and finally the base library "cmos".

When a piece of technology information is found in more than one library, the latest one is used (i.e. the current library's version is used before a dependent library's version, and a dependent library's version is used before that of another dependent library higher up the list). Note that the version which is used is expected to be the most recently created version, and a warning message will be issued if this is not the case.

Control of the library list is done with the **Edit Library Dependencies...** command (in menu **Edit / Technology Specific**).

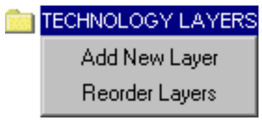
A dialog is presented with two lists of libraries. The list on the left shows the dependent libraries and the list on the right shows all current libraries. By selecting a library name from the list on the right and clicking on the "<< Add" button, it is added to the list on the left. To add a library not shown, type its name into the box on the right and click the "<< Add" button.



To remove a library from the list on the left, select it and click the "Remove" button.

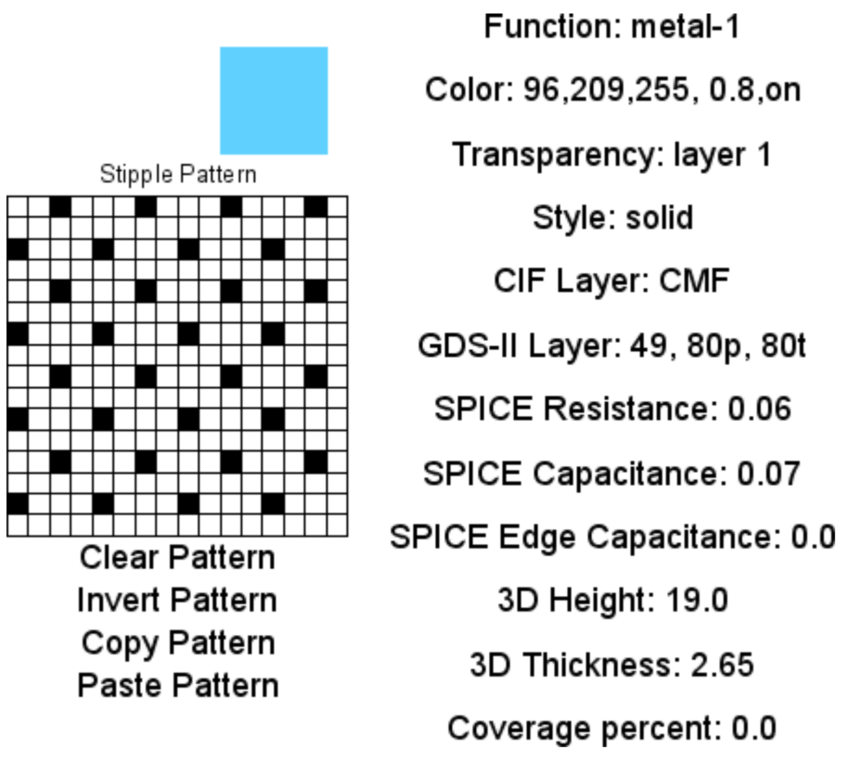
8−4: The Layers Cells

Layers are used to construct primitive nodes and arcs in a technology. Because of this, the layers must be edited before the nodes and arcs. To edit an existing layer, select it from the cell explorer or the **Edit Cell...** command (in menu **Edit**).



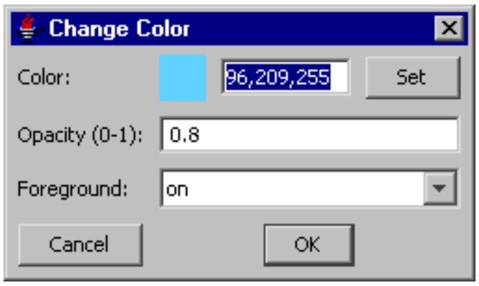
To create a new layer, use the context menu on the "TECHNOLOGY LAYERS" entry of the cell explorer and choose "Add New Layer". A layer can be deleted simply by deleting its cell. A layer can be renamed by renaming its cell, but remember to use the name "layer−" in front (i.e. the old name is "layer−metal" and the new name is "layer−metal−1"). Finally, you can rearrange the order in which the layers will be listed with the "Reorder Layers" command from the context menu.

There are many pieces of information in a layer, most of which can be updated by double−clicking on them. There is a 16x16 stipple pattern, a large square of color above that, and a number of pieces of textual information along the right side.



Function: metal-1
Color: 96,209,255, 0.8,on
Transparency: layer 1
Style: solid
CIF Layer: CMF
GDS-II Layer: 49, 80p, 80t
SPICE Resistance: 0.06
SPICE Capacitance: 0.07
SPICE Edge Capacitance: 0.0
3D Height: 19.0
3D Thickness: 2.65
Coverage percent: 0.0

The stipple pattern can be changed by double−clicking on any grid squares. You can also do operations on the entire stipple pattern ("Clear Pattern", "Invert Pattern", "Copy Pattern", and "Paste Pattern") by double−clicking on their name below the pattern area.



The color of the layer can be changed by double−clicking on the "Color" entry. The dialog lets you choose a color, opacity, and foreground factor for the layer. Opacity ranges from 1.0 (fully opaque) to 0 (transparent). The foreground flag is "on" to indicate that the non−opaque colors can be combined with others.

Transparency lets a layer have a unique appearance where it overlaps other layers. The overlap is defined in the technology's color map. You can double−click on the "Transparency" entry to assign this factor to a layer. Non−transparent layers (with "Transparency: none") are opaque, so they obscure anything under them when drawn. In general, the most commonly used layers should be transparent. See Section 4−6−1 for more information on transparency.

The "Style" entry on the right can be "solid", "patterned", or "patterned/outline" to indicate how that layer will be appear. When using "solid" styles, the 16x16 stipple pattern is ignored (except for hardcopy). The "patterned/outline" option draws a solid line around all patterned polygons. Transparent layers should be solid because they distinguish themselves in the color map. Layers with opaque colors should probably be patterned so that their combination is visible.

Many of the entries on the right side of the layer cell provide correspondences between a layer and various interchange standards. The "CIF Layer" entry is the string to use for CIF I/O. The "GDS−II layer" entry can be as simple as a single layer number, but it can also be two numbers separated by a "/" (the layer number and its type). You can also add a comma and then another layer/type pair with the letter "t" (for text) or "p" (for pin) at the end.

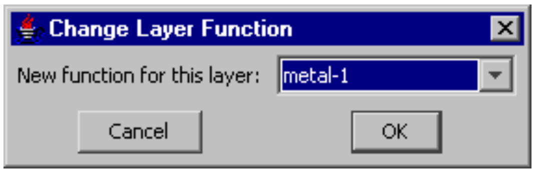
Another set of options on the right side of the layer cell is for Spice parasitics. You may assign a resistance, capacitance, and edge capacitance to the layer for use in creating Spice simulation decks.

The "3D Height" and "3D Thickness" are used when viewing a chip in 3−dimensions. The height and thickness are arbitrary values which describe the location and thickness in the third axis (out of the screen). For example, to show how poly and diffusion interact, the poly layer can be at height 21 and the diffusion layer at height 20, both with 0 thickness. This will appear as two ribbons, one over the other. See Section 4−11 for more information on 3D display.

The last option on the right side of the layer cell specifies the minimum coverage percentage (see Section 9−2−3 for more).

Layer Function

The "Function" entry allows a general−purpose description to be attached to the layer. A function consists of a single base description plus optional additional modifiers. The additional modifiers are found in the last entries of the function list.



These additional modifiers can be added to the base function:

- The modifiers "p−type," "n−type," "depletion," "enhancement," "light," "heavy", and "thick" describe layer types that are process−specific.
- The modifier "pseudo" indicates that this layer is a pseudo−layer, used for pin construction.
- The modifier "nonelectrical" indicates that this layer is decorative and not part of a real circuit.
- The modifiers "connects−metal," "connects−poly," and "connects−diff" indicate that this contact layer joins the specified real layers.
- The modifier "inside−transistor" indicates that the polysilicon is not field−poly, but is part of a transistor.

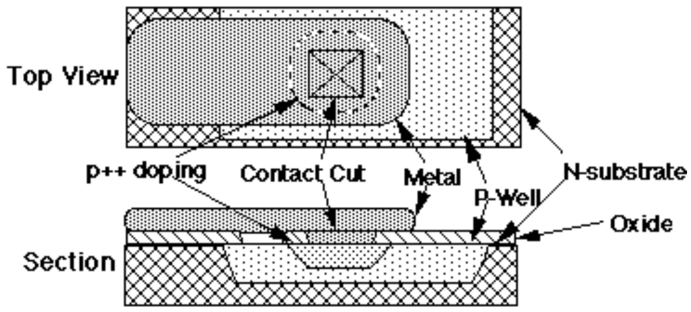
For example, you can double−click the function entry many times, selecting "Diffusion", "p−type", and "heavy" to indicate a Diffusion layer that is heavily−doped p−type. To clear the layer function, set it to "unknown."

A number of rules apply to the selection of layer functions. There must be a "pseudo" layer for every layer used to build arcs. This is because every arc needs a pin, and pins are constructed from "pseudo" layers. The "pseudo" layers are virtual geometry that do not appear in the fabrication output. It is important that every "pseudo" layer have an associated real layer, with similar descriptive fields. The technology editor will issue a warning if pins are not constructed from pseudo−layers.

Note that the layer functions must be treated carefully as they form the basis of subsequent arc and node definitions. One consideration to note is the use of "Wells" and their significance to the Spice extractor. If the technology requires a separate contact to the well, then it will typically contain a metal layer, and a piece of heavily doped material under the metal to make ohmic contact to the well; i.e. p++ in a P−well. This will have the same doping as the well, unlike a device diffusion, which is of opposite type to the well in which it is located.

Two rules apply here:

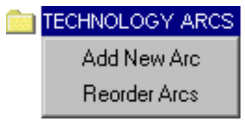
1. There must be a separate diffusion layer for the p++ or n++ used as a contact in a P−well or N−well, respectively; it cannot be the same layer that is used for diffusions in active devices.
2. A p++ or n++ layer that is used to make a contact in a well of the same semiconductor type (for example p++ in a P−well) must not be defined with the layer function Diffusion; it must be declared as "Well". In the well contact shown below, both the p++ layer and the P−well layer will be defined with the layer function "Well, P−type".



8−5: The Arc Cells

Creating and Deleting Arc Cells

Arcs are the wires in a technology, and they are constructed from pieces of geometry on the layers. To edit an existing arc, select it from the cell explorer or the **Edit Cell...** command (in menu **Edit**).



To create a new arc, use the context menu on the "TECHNOLOGY ARCS" entry of the cell explorer and choose "Add New Arc".

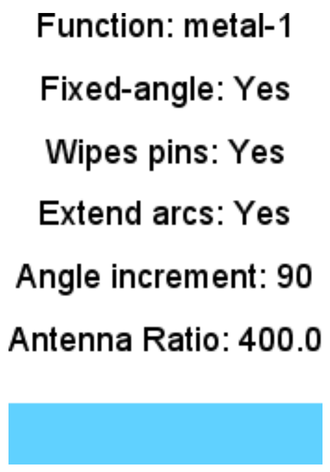
An arc can be deleted simply by deleting its cell. An arc can be renamed by renaming its cell, but remember to use the name "arc−" in front (i.e. the old name is "arc−metal" and the new name is "arc−metal−1"). Finally, you can rearrange the order in which the arcs will be listed with the "Reorder Arcs" command from the context menu.

Editing Special Arc Information

Arc cells show a sample arc on the bottom and a few pieces of textual information above it. The textual information can be updated by double−clicking on it.

The "Fixed−angle" entry lets you choose whether or not default arcs of this type are drawn at fixed angles (the particular fixed angle is specified by the "Angle increment" field below). In many layout technologies, the correct state is "yes".

The "Wipes pins" entry lets you choose whether or not these arcs completely erase connecting pins (the sensible state is "yes" because pins are drawn in the same layer and would not be visible anyway).



Function: metal-1

Fixed-angle: Yes

Wipes pins: Yes

Extend arcs: Yes

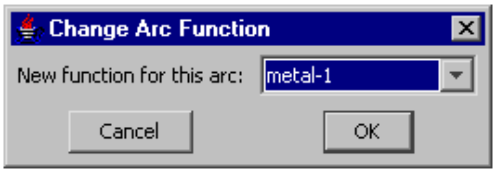
Angle increment: 90

Antenna Ratio: 400.0

The "Extend arcs" entry lets you choose whether or not these arcs extend beyond their endpoints by half of their width (the typical state is "yes").

The "Angle increment" entry is the preferred angle granularity of this type of arc (the typical state is "90" which requests Manhattan arcs).

The "Antenna Ratio" entry is used in antenna rules calculations (see Section 9−3−2).



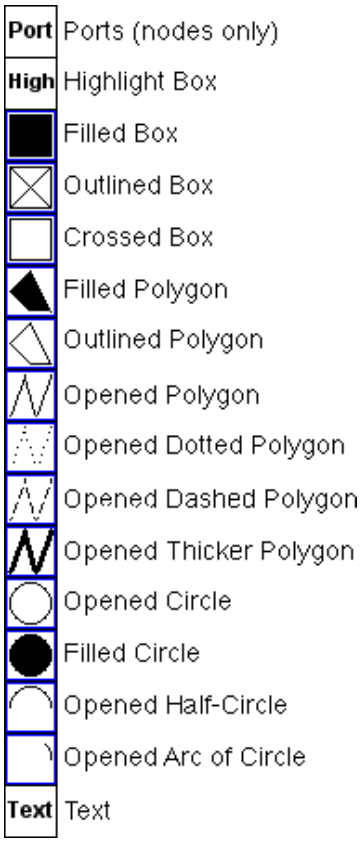
The "Function" entry describes the arc's function, which is a different set than the layer functions. As with layer functions, the arc functions should be carefully considered.

A well arc that contains a well layer and does not contain device diffusion (i.e. opposite doping to the well) must not be defined as "diffusion"; it must be defined as "well−diffusion". This prevents the Spice extractor from incorrectly adding any p or n doped area found in the well arc to the source or drain area of a transistor on the same network. This does not mean that a device arc cannot contain a well layer. Device arcs will be declared as "p−diffusion" or "n−diffusion", and their well layer will be handled correctly; the arc connectivity is really defined by the device diffusion layer. For example, a p−device arc will have an N−well, or N substrate under it, and a p−type diffusion will end up as part of the drain or source of the P transistor to which it is connected.

Editing Arc Geometry

In addition to the above information, the arc must also be described with pieces of geometry on the various layers. Thus, a prototypical arc must be drawn in this cell. The length of the arc is not important, but the smaller dimension is presumed to be the width and defines the default for this arc type.

Use the entries from the component menu of the side bar to create new layers. The typical layer in an IC technology is a Filled box (third from the top).



After the geometry is created, it can be moved and resized with standard Electric commands. Remember to keep all arc geometry separate from the information messages in the cell so that the technology editor can distinguish them. Once a piece of geometry is created, its layer can be set by double−clicking on it. A menu is then presented with possible layers (ignore the last entries, "SET−MINIMUM−SIZE", and "CLEAR−MINIMUM−SIZE" which are used only for nodes).

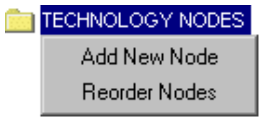
Besides geometric layers, the graphical arc description must have a highlight layer to show where the arc will be outlined when used in a circuit. Although the highlighting is typically drawn around the outside of all geometry, implant layers may extend beyond the highlight (see the CMOS diffusion arcs for an example of this). Select the "HIGH" entry in the component menu to create this special type of layer.

After geometry has been created, there may be some confusion as to what is there. To find out, use the **Identify Primitive Layers** command (in menu **Edit / Technology Specific**), which temporarily labels each piece of geometry in the arc cell.

8−6: The Node Cells

Creating and Deleting Node Cells

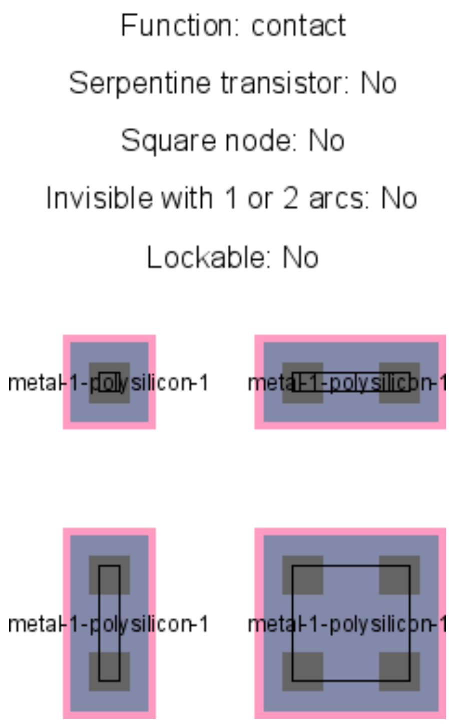
Nodes are the components in a technology, and they are constructed from pieces of geometry on the layers. To edit an existing node, select it from the cell explorer or the **Edit Cell...** command (in menu **Edit**).



To create a new node, use the context menu on the "TECHNOLOGY NODES" entry of the cell explorer and choose "Add New Node". A node can be deleted simply by deleting its cell. A node can be renamed by renaming its cell, but remember to use the name "node−" in front (i.e. the old name is "node−metal" and the new name is "node−metal−1"). Finally, you can rearrange the order in which the nodes will be listed with the "Reorder Nodes" command from the context menu.

Editing Special Node Information

The node cell contains four pictures of the node on the bottom and textual information above that. You can update the textual information entries by double−clicking on them.

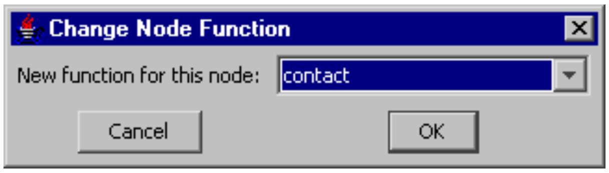


The "Serpentine transistor" entry indicates that this is a MOS transistor and it can take arbitrary outline information to describe its geometry.

The "Square" entry forces the node to always have the same X and Y dimension when scaled.

The "Invisible with 1 or 2 arcs" entry indicates that the node will not be drawn if it is connected to exactly one or two arcs. This is useful in schematic pins, which are visible only when unconnected or forming a junction of 3 or more wires.

The "Lockable" entry indicates that this node can be made unchangeable along with other lockable primitives, when the lock is turned on during editing (see Section 6−2 for more on locking these primitives). This is typically used in array technologies such as FPGA (see Section 7−6−2).



New function for this node: contact

The "Function" entry describes the node's function, which is a different set than the arc and layer functions. A dialog offers a list of possible node functions.

Editing Node Geometry

For nodes, it is common to sketch four different *examples* of the node in varying scales, so that X and Y scaling rules can be derived (square nodes need only two examples). If only one example is specified, default scaling rules will be presumed.

The smallest example, called the *main example*, is used as the default size and also contains all of the special port information. Needless to say, it is important to keep the geometry of each example well apart from the others so that the technology editor can distinguish them.

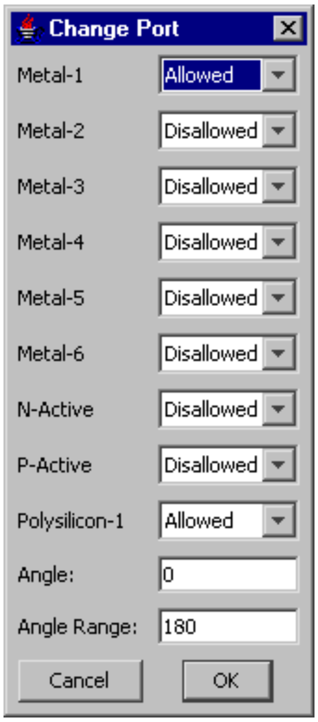
Each example must contain the same geometric layers (only stretched). As in the Arc cells, pieces of geometry can be created by selecting from the component menu of the side bar, creating the geometry, and then double−clicking to assign a layer. If any polygonal geometry is used (for example, the Filled polygon entry, sixth from the top), they require outline information to be assigned (see Section 6−10−1). If the Opened circle arc entry is selected (second from the bottom), you can specify the number of degrees of the circle with the **Object Properties...** command (in menu **Edit / Properties**).

Each example must also contain a highlight layer to indicate the correct highlighting on the display. Select the "HIGH" entry from the component menu to create this special type of layer.

Each example must also contain port information. Select the "PORT" entry in the component menu to create this special type of layer. You will have to provide a name for each port, and the name must be the same on each example.

Ports on the main example must also have connectivity information (which arcs can connect to them) and range information (the permissible angle of connected arcs). Double−click on the port to set this.

The range consists of two numbers: an angle (in degrees counterclockwise from 3 O'clock) and an angle range. For example, a port angle of 90 with a port angle range of 45 describes a port that points upward and can connect at angles up to 45 degrees off from this direction. The range will be graphically depicted.



Metal-1	Allowed
Metal-2	Disallowed
Metal-3	Disallowed
Metal-4	Disallowed
Metal-5	Disallowed
Metal-6	Disallowed
N-Active	Disallowed
P-Active	Disallowed
Polysilicon-1	Allowed
Angle:	0
Angle Range:	180

The ports on the main example must also indicate any internal electrical connectivity by actually connecting them together. For example, the two polysilicon ports on a MOS transistor should be connected in the main example. Join the ports with a universal arc. Do not put this internal connection on any example other than the main one. To see the location of all ports on the main example, use the **Identify Ports** command (in menu **Edit / Technology Specific**).

As with arcs, use the **Identify Primitive Layers** command to label each piece of geometry in the main example.

Special Node Considerations

There are some special cases available in node descriptions. A piece of geometry in the main example may be changed (by double−clicking on its function) to "SET−MINIMUM−SIZE". This indicates that the current size is the smallest possible, and it cannot scale any smaller (this is used by the "mocmos" technology for the metal layer in contacts). The restriction can be removed with the "CLEAR−MINIMUM−SIZE" description. This option cannot be used in serpentine transistors.

Another special case in node description is the ability to specify multiple cut layers. If the larger examples have more cut layers, rules are derived for cut spacing and indentation so that an arbitrary numbers of cuts can be inserted as the contact scales.

Although serpentine MOS transistors are a special case, they cannot be automatically identified, but must be explicitly indicated with a textual indicator. Besides this explicit indication, the transistor node must contain four ports: two on the gate layer (polysilicon) and two on the gated layer (active). A standard geometry must be used that shows polysilicon and diffusion crossing in a central transistor area. Any deviation from this format may cause the technology editor to be unable to derive serpentine rules for the node.

Besides the standard nodes for transistors, contacts, and other circuit elements, it is necessary to build pin and pure−layer nodes. There should be one pin for every arc, so that the arc can connect to others of its type. The pin should be constructed of pseudo−layers (i.e. it has no real geometry), should have the "pin" function, and should have one port in the center that connects to one arc. The technology editor will issue a warning if there is no pin node associated with an arc.

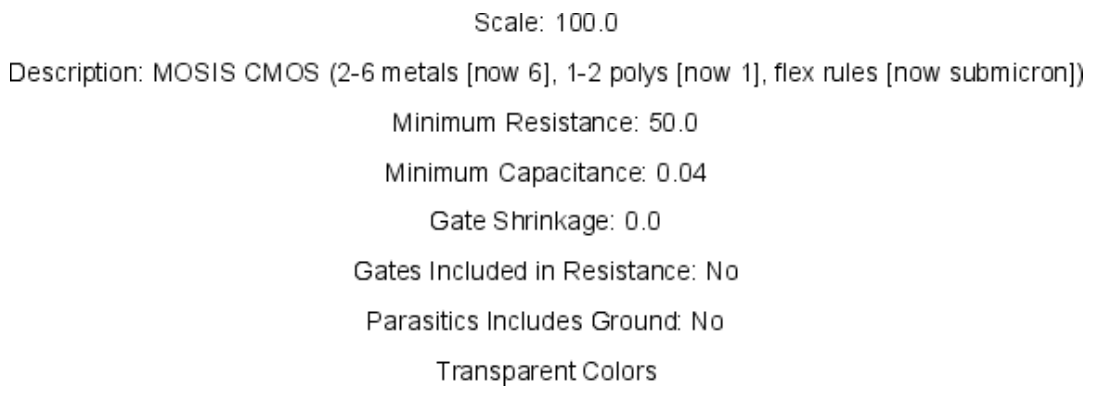
The pure−layer nodes should also be built, one for each layer. They should have only one piece of geometry and have the "pure−layer" function. The technology editor will issue a warning if there is no pure−layer node associated with a layer.

Chapter 8: Creating New Technologies

8−7: Miscellaneous Information

The Support Cell

Each cell in a technology library describes a different aspect of the technology. The *support* cell contains technology−wide information. To see this, edit the cell "factors" under the "TECHNOLOGY SUPPORT" section of the cell explorer.



Scale: 100.0

Description: MOSIS CMOS (2-6 metals [now 6], 1-2 polys [now 1], flex rules [now submicron])

Minimum Resistance: 50.0

Minimum Capacitance: 0.04

Gate Shrinkage: 0.0

Gates Included in Resistance: No

Parasitics Includes Ground: No

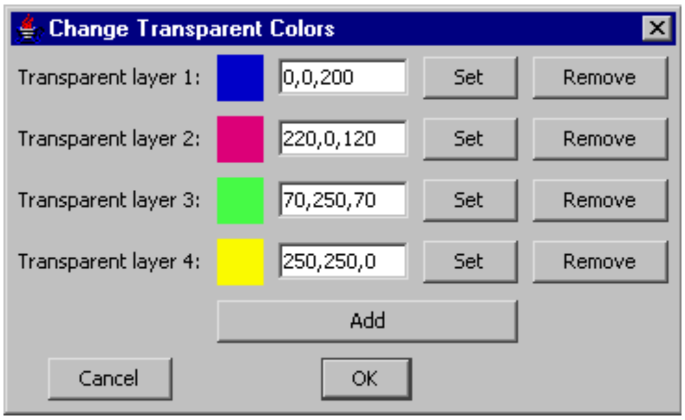
Transparent Colors

The support cell contains many items, any of which can be changed by double−clicking on it.

- "Scale" is the scaling factor between grid units and nanometers.
- "Description" is the full description of the technology.
- "Minimum Resistance" is the minimum resistance for the technology (see Section 9−10−1 for this and other parasitics).
- "Minimum Capacitance" is the minimum capacitance for the technology.
- "Gate Shrinkage" is the gate shrinkage for the technology.
- "Gates Included in Resistance" tells whether to include a transistor's gate in resistance computations.
- "Parasitics Includes Ground" tells whether to include ground networks in parasitics computations.

Transparent Colors

Double−clicking on the "Transparent Colors" entry shows a dialog for selecting the transparent colors. You must define as many colors as you have used in the layers.


Change Transparent Colors

Transparent layer 1:	🟦	0,0,200	Set	Remove
Transparent layer 2:	🟥	220,0,120	Set	Remove
Transparent layer 3:	🟩	70,250,70	Set	Remove
Transparent layer 4:	🟨	250,250,0	Set	Remove

Design Rules

Unfortunately, it is not possible to edit design rules associated with the technology.

8−8: How Technology Changes Affect Existing Libraries

Once a technology is created, the components are available for design. Soon there will be many libraries of circuitry that makes use of this new technology. What happens to these libraries when the technology description changes? In most cases, the change correctly affects the existing libraries. However, some changes are more difficult and might invalidate the existing libraries. This section discusses the possible changes and shows workarounds for the difficult situations.

Technology information appears in four different places: the layers, the arcs, the nodes, and miscellaneous information on the technology (the support cell and color tables). Information in these areas can be added, deleted, or modified. The rest of this section outlines all of these situations.

Adding layers, adding arcs, adding nodes, adding miscellaneous information

Adding information has no effect on the existing circuitry. All subsequent circuit design may make use of the new technology elements.

Deleting layers

All references to a deleted layer, in any nodes or arcs of the technology, will become meaningless. This does not invalidate libraries that use the layers, but it does invalidate the node and arc descriptions in the technology. The geometry in these nodes and arcs will have to be moved to another layer.

Deleting nodes, deleting arcs

This will cause error messages when libraries are read that make use of the deleted elements. When the library is read, you can substitute another node or arc to use in place of the now−unknown component.

Deleting miscellaneous information

This depends entirely on where that information is used. For example, an analysis tool may fail to find the information that it requires.

Modifying layers

This is a totally transparent operation. Any change to the color, style, or stipple information (including changes to the color map) will appear in all libraries that use the technology. Changes to I/O equivalences or Spice parasitics will be available to all existing libraries. A change of the layer function may affect the technology editor's ability to decode the nodes and arcs that use this layer (for example, if you change the function of the "polysilicon" or "diffusion" layers that form a transistor, the editor will be unable to identify this transistor). Renaming a layer has no effect.

Modifying arcs, modifying nodes

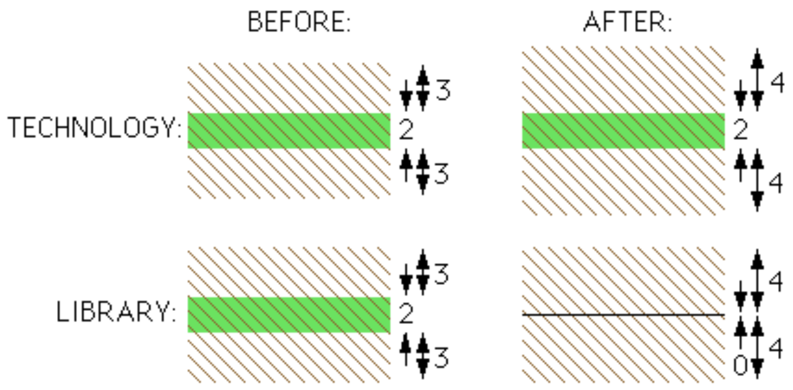
This is not as simple as layer modification because the arcs and nodes appear in the circuit libraries, whereas the layers do not. If you rename a node or arc, it will cause errors when libraries are read that make use of nodes with the old name. Therefore, you must create a new node or arc first, convert all existing ones to the new type, and then delete the old node or arc.

Many of the pieces of special information on the top of the node and arc cells apply to newly created circuitry only, and do NOT affect existing components already in libraries. The arc factors "Fixed−angle", "Wipes pins", "Extend arcs", and "Angle increment" have no effect on existing libraries. The node factor "Square node" also has no effect on existing circuitry and gets applied only in subsequent designs.

Other factors do affect existing circuitry. Changes to the "Function" field, in both arcs and nodes, pass to all existing components, thus affecting how analysis tools treat the old circuits. If the "Serpentine Transistor" field in nodes is turned off, any existing transistors that have serpentine descriptions will turn into large rectangular nodes with incorrect connections (i.e. get trashed). Unfortunately, it may become impossible to keep the "Serpentine Transistor" field on if the geometry does not conform to standards set by the technology editor for recognizing the parts. If a node is not serpentine, turning the factor on has no effect. Finally, the node factors "Invisible with 1 or 2 arcs" and "Lockable" correctly affect all existing circuitry.

A more common modification of arcs and nodes is to change their graphical descriptions. A simple rule applies to all such changes: if you change the size of the bounding box, it causes possibly unwanted proportion changes in all existing circuitry. This is because the bounding box information is all that is stored in the library, and layer sizes are defined in terms of that box.

For example, assume that there is an active arc defined with two layers: diffusion (2 wide) and well (8 wide). The arcs in the libraries are therefore recorded as being 8 wide (the largest size). The system knows that the diffusion is narrower than the overall arc by 3 on each side.



Now, if you change the well so that it is 10 wide, the system will define the diffusion to be narrower than the overall arc by 4 on each side, and for the existing 8−wide arcs, the diffusion will shrink to zero and disappear. These arcs must be resized individually, which can be tedious.

Here is an example of how node geometry changes can also make trouble. Assume that there is a transistor that has an active piece (2 wide) and a gate piece (2 wide). Each piece extends beyond the transistor area by 2, thus making the entire node 6x6 in size. The size of each cross piece will be defined to be 2 narrower than the bounding box on each side. If the pieces are changed so that they extend by only 1, then the definition of each strip will change to being 1 less than the box size on each side. All existing 6x6 transistors will suddenly have 4x4 gate areas where they used to be 2x2.

In both of these examples, it may be preferable to keep the old technology and give the new technology a different name. Then the old libraries can be read into the old technology, and the **Make Alternate Layout**

View... command (in menu **View**) can be used to translate into the new technology. This command uses node and arc functionality to associate components, scaling them appropriately relative to their default sizes. The change is completed by deleting the old technology, renaming the new technology to the old name, and then saving the library.

Modifying miscellaneous information

This last situation is typically transparent: changed information appears in all existing libraries, and affects those subsystems that make use of the information. For example, a change to the Spice resistance will be seen when a Spice deck is next generated.

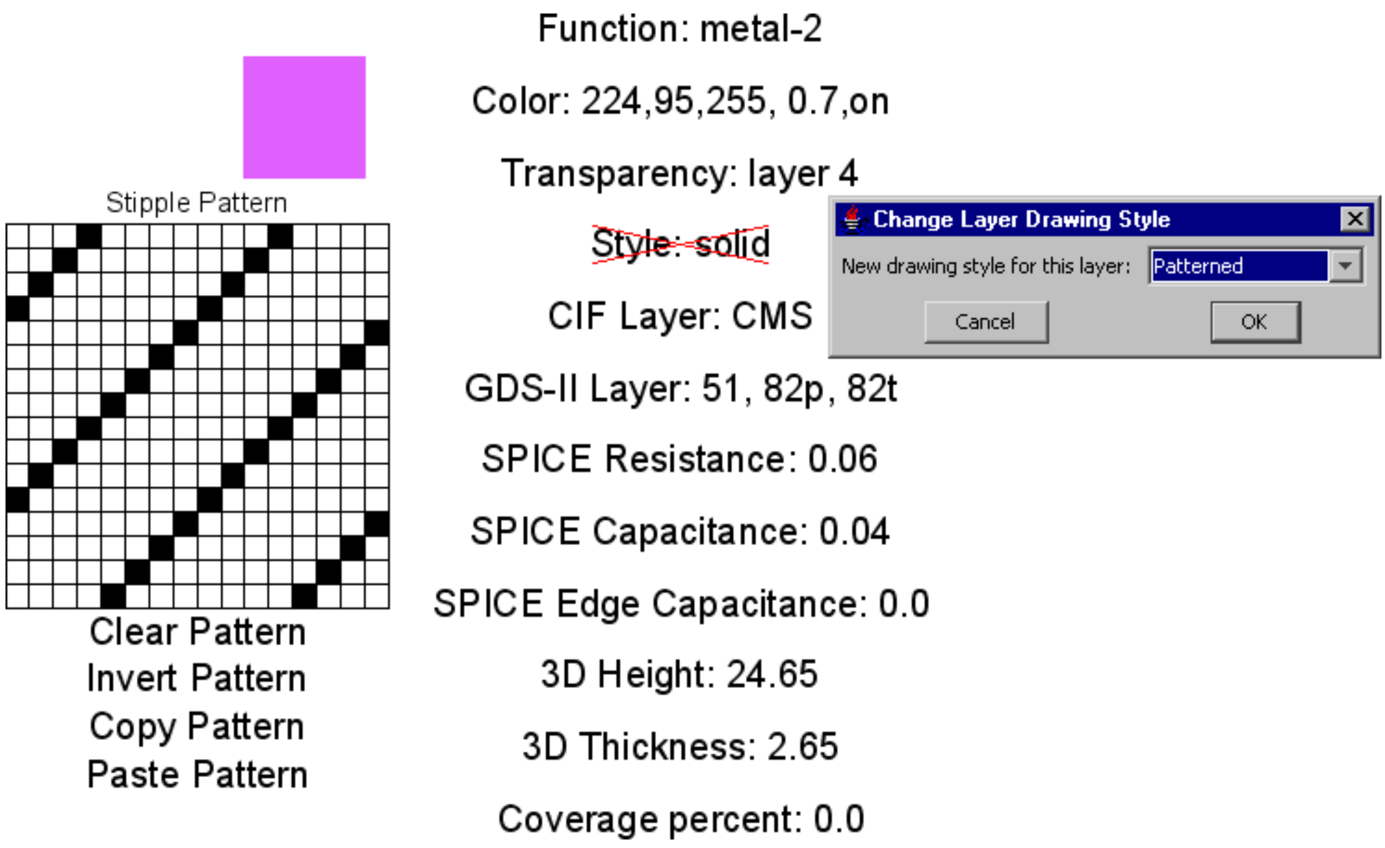
8−9: Examples of Use

To fully understand technology editing, some examples are appropriate. Two examples will be given: a simple one that modifies the appearance of a pattern, and a more complex example in which a new primitive node is created. Both examples are based on the MOSIS CMOS technology, so they presume that the **Convert Technology to Library for Editing...** command (in menu **Edit / Technology Specific**) has been issued and the "mocmos" entry was selected.

Example: Modifying a Layer's Appearance

In this first example, the user simply wishes to change the Metal−2 layer from a solid fill to a stipple pattern.

This particular task is so basic that it can be done with the "Layers" preferences, but it illustrates the basic steps of making a change. First, edit the layer cell for "metal−2". The display will show the layer with all of its associated information.



Function: metal-2
Color: 224,95,255, 0.7,on
Transparency: layer 4
~~Style: solid~~
CIF Layer: CMS
GDS-II Layer: 51, 82p, 82t
SPICE Resistance: 0.06
SPICE Capacitance: 0.04
SPICE Edge Capacitance: 0.0
3D Height: 24.65
3D Thickness: 2.65
Coverage percent: 0.0

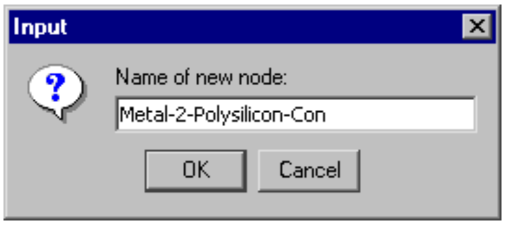
New drawing style for this layer: Patterned

Because every layer has a default stipple pattern used for printing, all that is necessary is to change the "Style" field from solid to patterned. To do this, double−click on the "Style" text and select "Patterned". The technology is now modified and can be converted back with the **Convert Library to Technology...** command.

Example: Creating a New Node

The second example is more extensive: creation of a new primitive node. In this case, the new node is a contact between metal−2 and polysilicon.

To create the node, use the context menu on the "TECHNOLOGY NODES" tab of the explorer window, select "Create New Node", and name the node appropriately.


Input

Name of new node: Metal-2-Polysilicon-Con

At this point, the display will show only the textual information about the node (because the graphical information is yet to be supplied). The textual information consists of five factors that now fill the screen.

Function: contact

Serpentine transistor: No

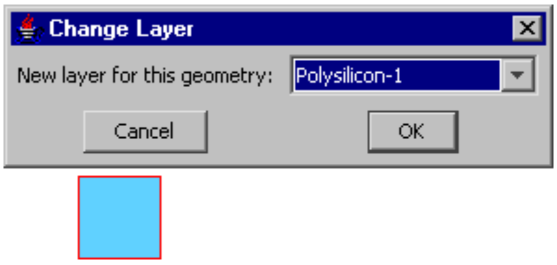
Square node: No

Invisible with 1 or 2 arcs: No

Lockable: No

You should begin by changing the "Function" factor to "contact" (double−click it and select the appropriate function). Then pan back so there is room to describe the node graphically. The other factors are properly set for a contact.

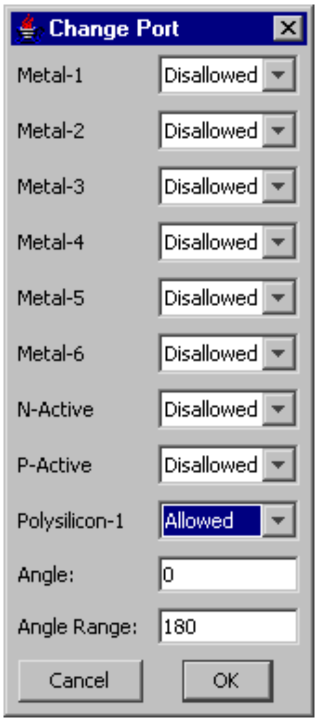
To place a piece of geometry (for example, some polysilicon), click over the filled box entry in the component menu (third from the top) and then click in the edit window. This geometry now has shape, but no layer associated with it. To assign a layer, double−click on the geometry. Then choose "polysilicon−1". The black box will change appearance to that of a polysilicon layer. You can move and stretch this box appropriately.


Change Layer

New layer for this geometry: Polysilicon-1

In this example, assume that a contact between polysilicon and metal−2 has three layers: polysilicon−1, metal−2, and contact cut. Therefore, the above operation must be done two more times to place the metal−2 and contact cut layers.

Besides this pure geometry, there must be two other items in the node: a highlight layer and a port. The highlight layer is obtained by selecting the "HIGH" entry from the component menu. It is then placed and stretched so that it encloses the contact (highlight layers define the size of the node, and this means that they will typically surround the geometry).



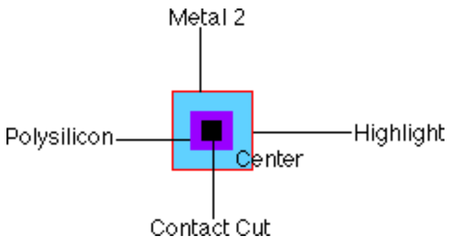
Change Port

Metal-1	Disallowed
Metal-2	Disallowed
Metal-3	Disallowed
Metal-4	Disallowed
Metal-5	Disallowed
Metal-6	Disallowed
N-Active	Disallowed
P-Active	Disallowed
Polysilicon-1	Allowed
Angle:	0
Angle Range:	180

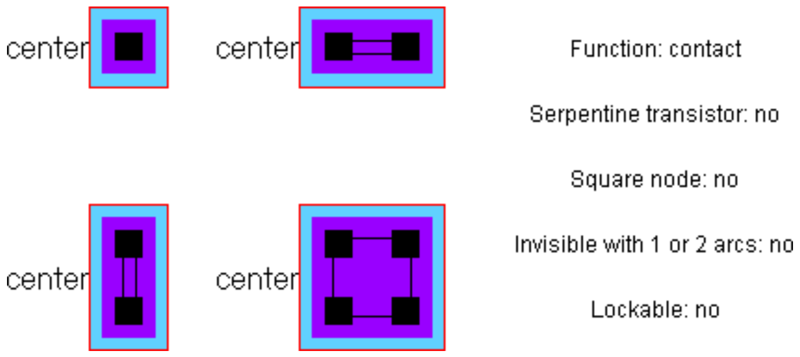
The other item that must be created is a port (more than one can be created, but for contacts, one is sufficient). Select the "PORT" entry from the menu on the left and place it in the display. You will be prompted for a port name, after which you can further move or stretch the port. Besides a location and a name, ports must specify which arcs may connect to them. To do this, double−click on the port.

The resulting menu lists all of the arcs and indicates possible connectivity. Note that the last two entries define the permissible range of angles to which arcs may connect. For a contact such as this, arcs may connect at any angle, so the default values are correct.

When all of the geometry, highlighting, and ports have been placed, you can double−check your work with the **Identify Primitive Layers** command (in menu **Edit / Technology Specific**), which will display this information (note that the port name "Center" has been moved away for clarity):



The final step in the definition of this node is to create three more copies that illustrate scaling in both axes. This is done simply by selecting all five objects and using the **Duplicate** command (in menu **Edit**). Once duplicated in a new location, each piece must be stretched appropriately. In this example, the contact cut is designed so that the number of cut elements grows with the node. Thus, when stretched horizontally or vertically, there are two cuts, and when stretched in both directions there are four cuts. The technology editor will determine precise multicut rules from the cut spacing and the amount of stretch, so that even more cuts will appear as the node grows larger. The finished node definition is shown below:



Function: contact

Serpentine transistor: no

Square node: no

Invisible with 1 or 2 arcs: no

Lockable: no

All that is necessary is to convert this library back to a technology, and the new technology will have this node.

Of course, the newly created technology is valid only during the current session. Therefore, to preserve this technology, save the library to disk. In subsequent sessions, you must read the technology library and convert it to a technology before using it. Alternatively, you can request the system to write Java code for the technology and compile it into Electric..

Chapter 9: Tools

9−1: Introduction to Tools

There are many different tools available in Electric for doing both synthesis and analysis of circuitry. Synthesis tools include routers, compactors, circuit generators, and so on. Analysis tools include design−rule checkers, network comparison, and many simulators. To see a list of tools, including which ones are active, use the **List Tools** command (in menu **Tool**). This chapter covers many of the tools available in Electric.

When a tool is running, it may take a long time. You can see it under the "JOBS" entry of the cell explorer (see Section 4−8).

After a tool has run, it may reports errors in the ERRORS section of the cell explorer. To browse these errors, use the **Show Next Error** and **Show Previous Error** commands (in menu **Edit / Selection**) or type the ">" an "<" keys.

A number of common tool controls are available from the "General" preferences (in menu **File / Preferences...**, "General" section, "General" tab), especially in the "I/O" and "Jobs" section.

Most netlisters insert date and version information in the comments at the head of the generated file. You can request that this information be omitted by unchecking "Include date and version in output files".

Most of the commands to generate an input deck for a simulator (a netlist) prompt the user for the desired file. If "Show file−selection dialog before writing netlists" is unchecked, however, the file is written (or overwritten) without prompt. This is useful in repetitive iterations of design/simulate, and saves the cumbersome file−selection dialog. However, it can be dangerous because it overwrites files without asking.

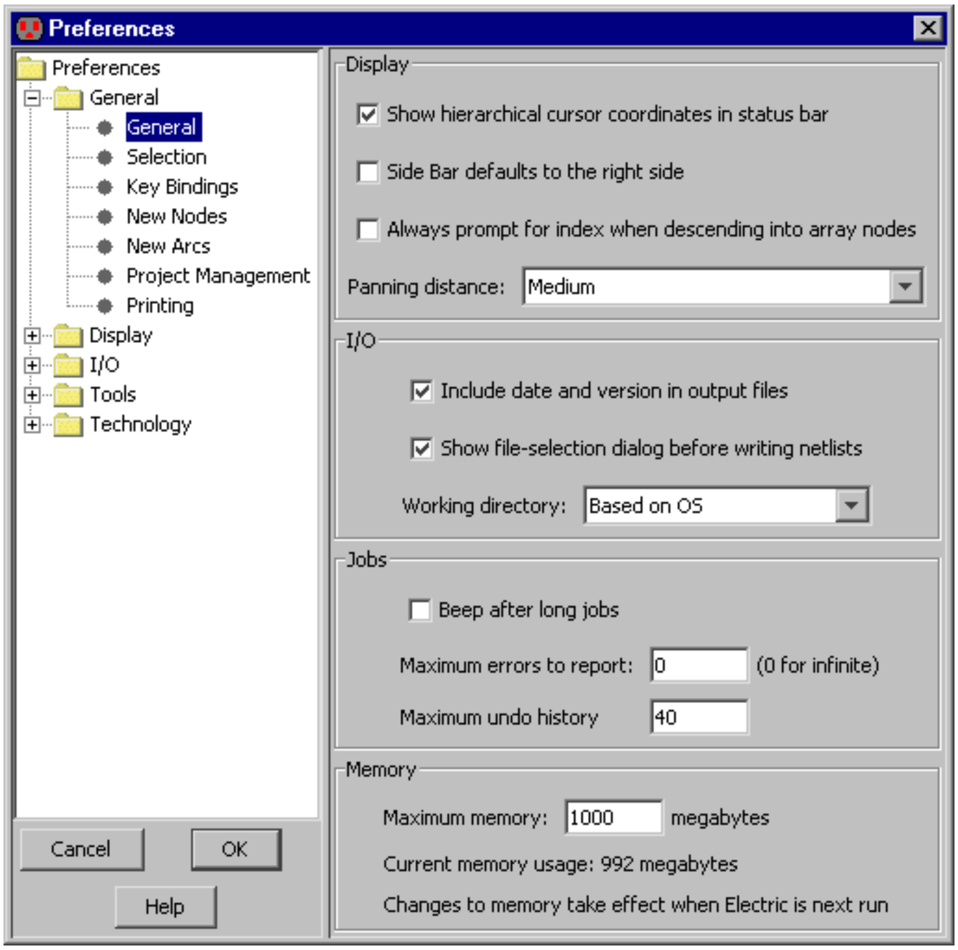
For more information about "Working directory", see Section 3−9−1.

In the "Jobs" section, "Beep after long jobs" requests that any job which runs longer than a minute make a beep sound when done.

You can set the maximum number of errors that will be reported at once. By default, there is no limit to the number of errors.

For more information about "Maximum undo history", see Section 6−7.

For more information about the "Display" section, see Section 1−7 ("Show hierarchical cursor coordinates in status bar"), Section 4−8 ("Side Bar defaults to the right side"), Section 3−5 ("Always prompt for index when descending into array nodes"), and Section 4−4−2 ("Panning distance").



Display

[✓] Show hierarchical cursor coordinates in status bar

[ ] Side Bar defaults to the right side

[ ] Always prompt for index when descending into array nodes

Panning distance: Medium

I/O

[✓] Include date and version in output files

[✓] Show file-selection dialog before writing netlists

Working directory: Based on OS

Jobs

[ ] Beep after long jobs

Maximum errors to report: 0 (0 for infinite)

Maximum undo history 40

Memory

Maximum memory: 1000 megabytes

Current memory usage: 992 megabytes

Changes to memory take effect when Electric is next run

For more information about the "Memory" section, see Section 1−3.

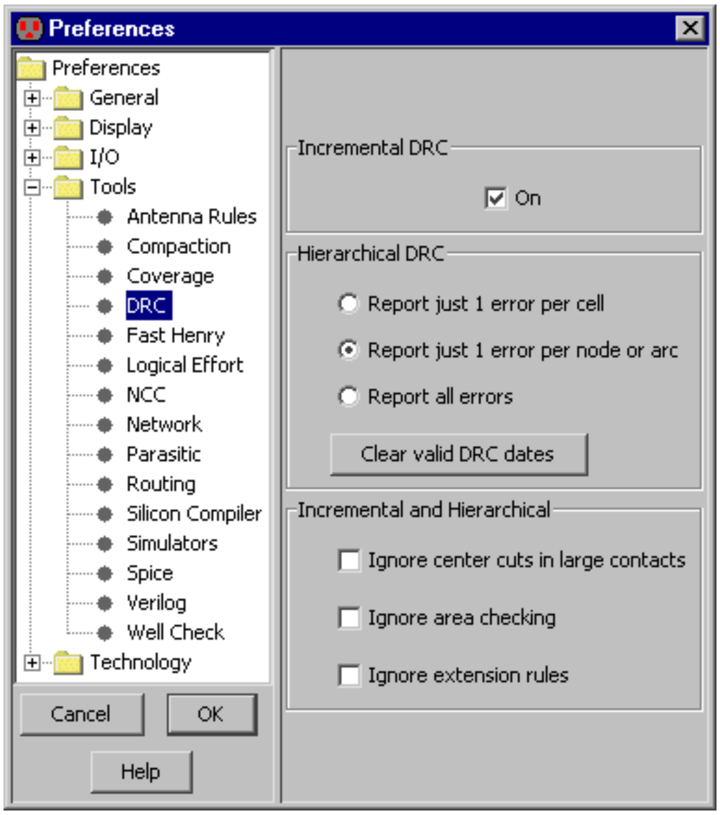
9−2−1: Incremental DRC

The incremental design−rule checker is always running, examining your work, and issuing error messages when an error is detected.

To control the DRC, use the "DRC" preferences (in menu **File / Preferences...**, "Tools" section, "DRC" tab).

By default, the incremental design−rule checker is on. To turn it off, uncheck the "On" checkbox in the "Incremental DRC" section.

MOS contact nodes automatically increase the number of cuts when they grow larger (see Section 7−4−1). Because of this, very large contact nodes can create excessive work for the design−rule checker as it examines each of the cuts. To save time, check the "Ignore center cuts in large contacts" check box, which will examine only the cut layers around the edges of contact nodes.



Incremental DRC

[✓] On

Hierarchical DRC

( ) Report just 1 error per cell
(●) Report just 1 error per node or arc
( ) Report all errors

Incremental and Hierarchical

[ ] Ignore center cuts in large contacts
[ ] Ignore area checking
[ ] Ignore extension rules

DRC rules for new technologies might require special rules, which can be time consuming. To ignore these errors, check "Ignore area checking" (for minimum area rules) and "Ignore extension rules" (for special overlap rules).

After analysis of the circuit, you can review the errors by typing ">" and "<" to step to the next and previous error that was found. You can also see a list of errors in the cell explorer (see Section 4−8).

9−2−2: Hierarchical DRC

The hierarchical design−rule checker uses the same rules and techniques as the incremental checker, but it checks all levels of hierarchy below the current cell. To run it, use the **Check Hierarchically** command (in menu **Tool / DRC**). To check only a selected subset of the current cell, use **Check Selection Area Hierarchically**.

After analysis of the circuit, you can review the errors by typing ">" and "<" to step to the next and previous error that was found. You can also see a list of errors in the cell explorer (see Section 4−8).

After a cell has passed Hierarchical DRC with no errors, it is tagged with the current date. In subsequent runs of the Hierarchical DRC, if the cell has not been modified since that date, it is not rechecked. (However, if you change the DRC rules or the technology options, all date information is cleared.) If you wish to force all cells to be rechecked, use the "Clear valid DRC dates" button in the "DRC" preferences (in menu **File / Preferences...**, "Tools" section, "DRC" tab). To see which cells have passed Hierarchical DRC, use the **General Cell Lists...** command (in menu **Cell / Cell Info**) A "D" is shown in on the right for cells that are DRC current (see Section 3−7−1).

There are three levels of checking that can be requested in the "DRC" preferences, each consuming more time and finding more errors.

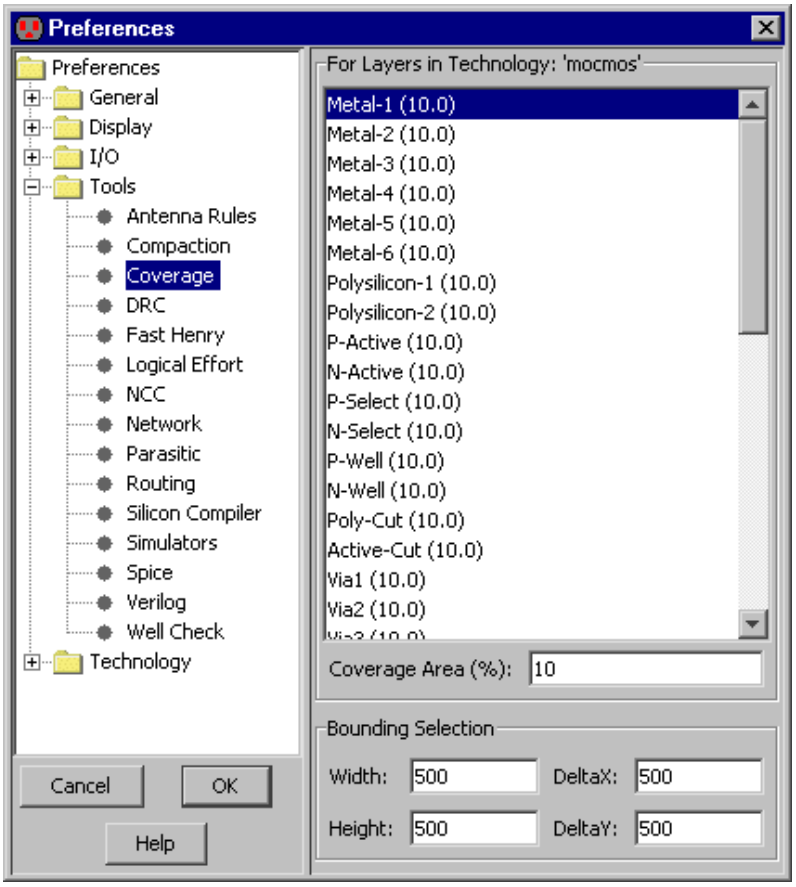
- "Report just 1 error per cell" tells the system to stop checking a cell after the first error has been found. By using this option, you can more quickly determine *which* cells in the design are correct, without knowing exactly where the errors lie. Then, you can go to the cells with errors and do a more complete check.
- "Report just 1 error per node or arc" is the default. It stops checking a node or arc when it has found any design rule violation.
- "Report all errors" tells the system to continue checking a node or arc, even if an error has been found. This will report all violations found, but can take more time.

9−2−3: Coverage Rules

Some foundries request that each layer occupy a minimum percentage of the chip. To enforce such rules, additional pieces of geometry must be placed around the chip to fill that layer.

To check for proper minimum layer coverage, use the **Check Area Coverage** command (in menu **Tool / DRC**). To control the coverage rules, use the "Coverage" preferences (in menu **File / Preferences...**, "Tools" section, "Coverage" tab). Each layer in the technology has a minimum percentage of coverage that is needed.

The coverage check proceeds in a "tiled" manner, checking rectangular areas of the cell. For example, to check each 100x100 unit area of the cell, set "Width" and "Height" to 100, and set "DeltaX" and "DeltaY" to 100.



For Layers in Technology: 'mocmos'

Metal-1 (10.0)
Metal-2 (10.0)
Metal-3 (10.0)
Metal-4 (10.0)
Metal-5 (10.0)
Metal-6 (10.0)
Polysilicon-1 (10.0)
Polysilicon-2 (10.0)
P-Active (10.0)
N-Active (10.0)
P-Select (10.0)
N-Select (10.0)
P-Well (10.0)
N-Well (10.0)
Poly-Cut (10.0)
Active-Cut (10.0)
Via1 (10.0)
Via2 (10.0)

Coverage Area (%): 10

Bounding Selection

Width:	500	DeltaX:	500
Height:	500	DeltaY:	500

The **List Layer Coverage on Cell** command is another way to compute the percentage of the cell that is covered by each layer. This command covers the entire cell without breaking it into tiled rectangles.

9−2−4: Assura DRC

Electric is able to read the output of Cadence's Assura design−rule checker. These error files (with the extension ".err") can be read with the **Import Assura DRC Errors...** command (in menu **Tool / DRC**). After reading the error file, you can review the errors by typing ">" and "<" to step to the next and previous error that was found. You can also see a list of errors in the cell explorer (see Section 4−8).

9−2−5: Design Rules

Four types of errors are detected by the incremental and hierarchical design−rule checkers. *Spacing* errors are caused by geometry that is too close, but not connected. *Notch* errors are caused by geometry that is too close, but connected. *Minimum size* errors are caused by geometry that is too small. *Resolution* errors are caused by geometries that are smaller than a requested resolution amount.

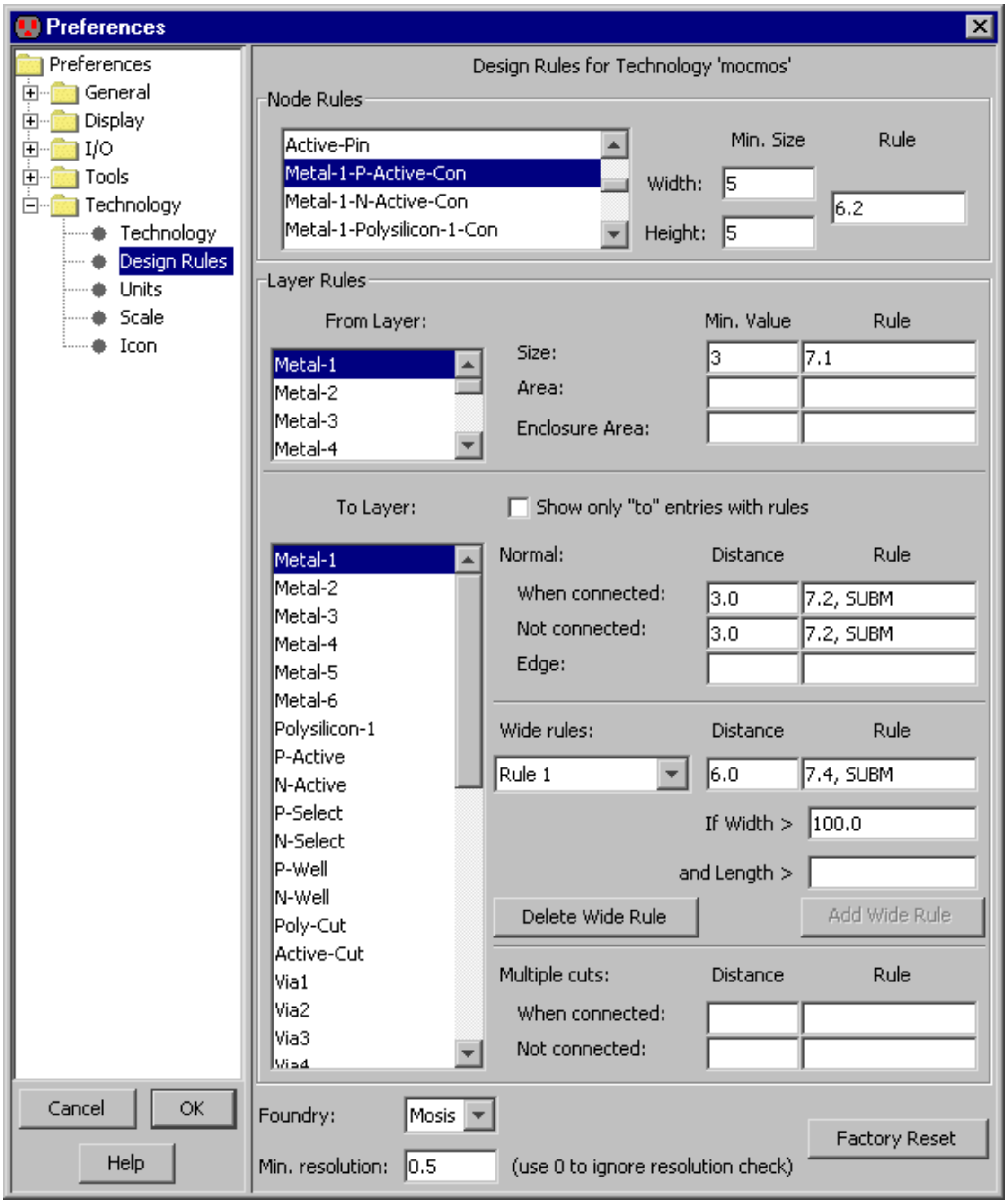
In addition to examining geometry, the design−rule checkers use connectivity information to help find violations. This use of network information helps the designer to debug circuit connectivity. For example, if two overlapping nodes are not joined by an arc, they may be considered to be in violation, even if their geometry looks right. This is because the checkers know what is connected and have a separate set of rules for such situations.

To help guide the design−rule checker, a "cloaking" layer can be placed over areas that are not to be examined. This cloaking layer is created by clicking the "Misc." entry of the component menu and selecting "DRC Exclusion". Any errors that fall inside of this node's area are ignored.

To edit the design rules, use the "Design Rules" preferences (in menu **File / Preferences...**, "Technology" section, ""Design Rules" tab). The dialog allows you to examine and modify the spacing limits for the current technology. Each rule has a numeric value (size or distance) as well as a textual description of the rule. The dialog is divided into two parts: "Node Rules" and "Layer Rules".

In the "Node Rules" section, you may set the minimum size of each node in the current technology.

In the "Layer Rules" section, you may set the minimum size, area, and enclosure area of each layer. You may also set the inter−layer spacing (between the "From Layer" and the "To Layer"). Use the "Show only 'to' entries with rules" to restrict the displayed rules to those with valid values.



Design Rules for Technology 'mocmos'

Node Rules

Active-Pin
Metal-1-P-Active-Con
Metal-1-N-Active-Con
Metal-1-Polysilicon-1-Con

	Min. Size	Rule
Width:	5	6.2
Height:	5	

Layer Rules

From Layer: Metal-1, Metal-2, Metal-3, Metal-4

	Min. Value	Rule
Size:	3	7.1
Area:		
Enclosure Area:		

To Layer: Metal-1, Metal-2, Metal-3, Metal-4, Metal-5, Metal-6, Polysilicon-1, P-Active, N-Active, P-Select, N-Select, P-Well, N-Well, Poly-Cut, Active-Cut, Via1, Via2, Via3, Via4

☐ Show only "to" entries with rules

Normal:	Distance	Rule
When connected:	3.0	7.2, SUBM
Not connected:	3.0	7.2, SUBM
Edge:		

Wide rules:	Distance	Rule
Rule 1	6.0	7.4, SUBM

If Width > 100.0

and Length >

Multiple cuts:	Distance	Rule
When connected:		
Not connected:		

Foundry: Mosis

Min. resolution: 0.5 (use 0 to ignore resolution check)

The layer−to−layer spacing rules appear in 3 forms: *normal*, *wide*, and *multicut*. Normal and multicut rules come in two flavors: connected and unconnected. The connected rules apply to pieces of geometry that are electrically connected; the unconnected rules apply to unconnected geometry. The normal rules also have a special Edge rule applies only to unconnected layers and ignores overlap when considering spacing distance.

The wide rules apply to large geometry. Although some technologies may have many different rules for different definitions of "large", the MOSIS CMOS technology has only one such rule, if the width is greater than a specified value.

The bottom of the dialog has some special features of design rules:

- The "Factory Reset" button restores all rules to the original set built into Electric.
- The "Foundry" controls which foundry is the target (these are variations on the design rules). Currently, this is not used.

- The "Min resolution" is the minimum resolution that can be manufactured. If zero, no resolution check is done. When checking resolution, all geometry of that size or less will be flagged as resolution errors. For example, current MOSIS rules require that no boundaries be quarter−lambda or less, so a value of .25 in this field will detect such violations.

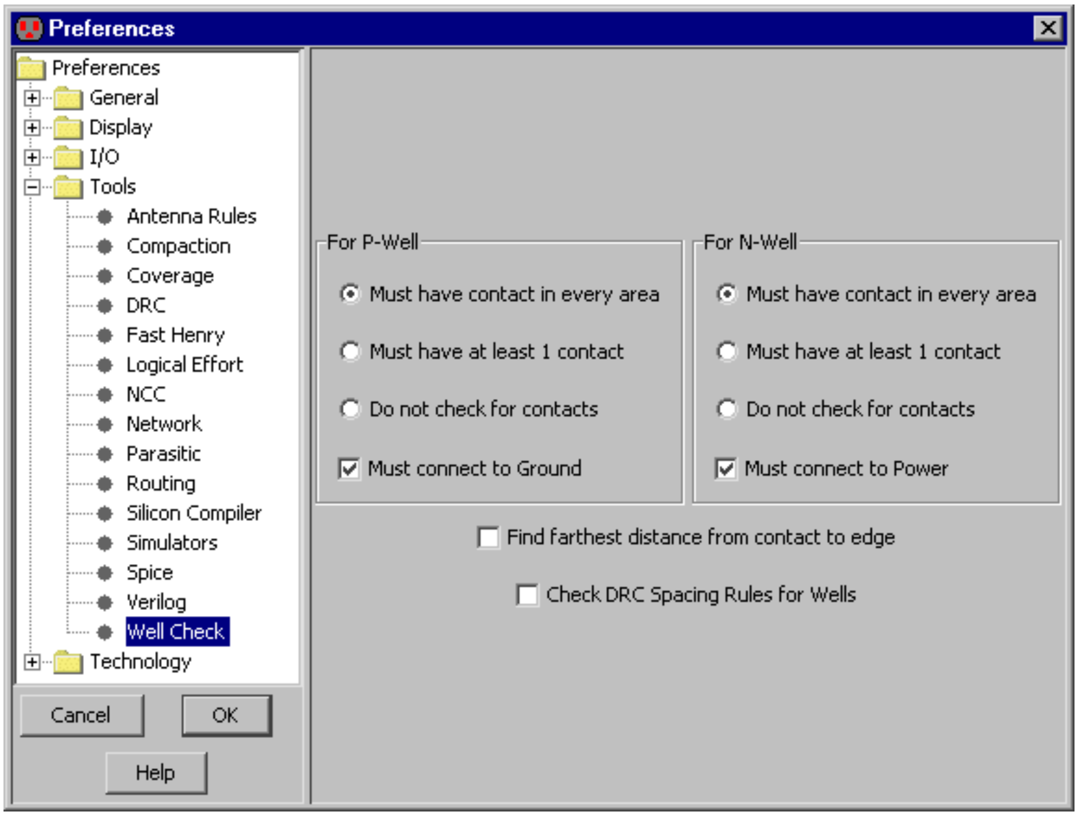
Note that the MOSIS CMOS design rule 6.7b is not checked by Electric because it is difficult to detect properly. This error is never fatal, and the worst case of missing this error is that active and poly are closer by 1/2 lambda, which merely results in an increase in capacitive coupling between them. If this fringing capacitance is important, you've probably got so much polysilicon in your circuit that it has bigger problems.

9−3−1: Well and Substrate Checking

To check the well and substrate layers, use the **Check Wells** command (in menu **Tool / ERC**). This does a more thorough job of checking the layers than the design−rule checker.

After analysis is done, you can review the errors by typing ">" to see the next error and "<" to see the previous error. You can also see the list of errors in the cell explorer (see Section 4−8).

You can control the Well Checker with the "Well Check" preferences (in menu **File / Preferences...**, "Tools" section, "Well Check" tab).



For P-Well
[✓ Must have contact in every area]
Must have at least 1 contact
Do not check for contacts
[✓ Must connect to Ground]

For N-Well
[✓ Must have contact in every area]
Must have at least 1 contact
Do not check for contacts
[✓ Must connect to Power]

Find farthest distance from contact to edge
Check DRC Spacing Rules for Wells

The Well Checker makes sure that there are well contacts in every area of well. The dialog allows you to relax that restriction and demand only 1 well contact in each cell, or not to check for contacts at all.

The Well Checker also checks that there is a connection to power and ground in the appropriate places. You can disable these checks in the "Well Check" dialog.

An additional well check is to find the farthest distance from a substrate contact to the edge of that area. This check takes more time to do, and so it can be disabled.

The Well Checker can check spacing rules between well areas. Although this is generally the domain of the Design Rule Checker (DRC), it could be done here by checking "Check DRC Spacing Rules for Wells". Since the well checker has not been designed for DRC purposes, the algorithm is not efficient and therefore the option is off by default.

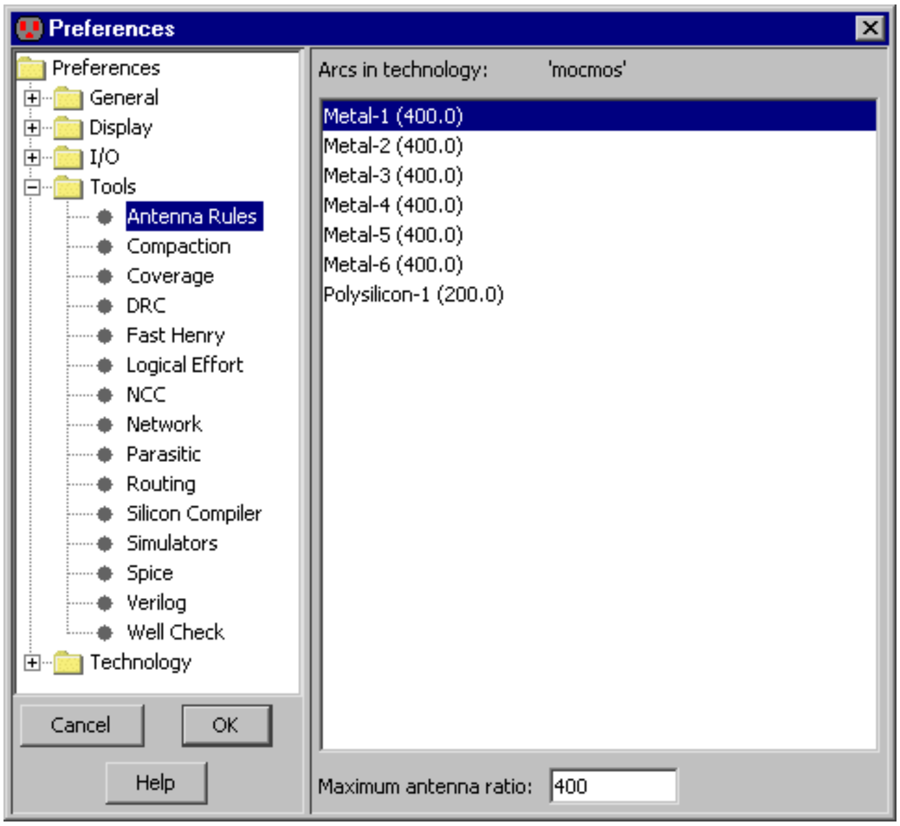
Finally, the Well Checker reports the maximum distance from a well contact to any point on the well. This is useful when making sure that there are sufficient contacts for each area.

9−3−2: Antenna Rule Checking

Antenna rules are required by some IC manufacturers to ensure that the transistors of the chip are not destroyed during fabrication. In such processes, the wafer is bombarded with ions in order to create the polysilicon and metal layers. These ions must find a path to through the wafer (to the substrate and active layers at the bottom). If there is a large area of poly or metal, and if it connects ONLY to gates of transistors (not to source or drain or any other active material) then these ions will travel through the transistors. If the ratio of the poly or metal layers to the area of the transistors is too large, the transistors will be destroyed.

To check for antenna rule violations, use the **Antenna Check** command (in menu **Tool / ERC**). After analysis is done, you can review the errors by typing ">" to see the next error and "<" to see the previous error. You can also see the list of errors in the cell explorer (see Section 4−8).

You can control the Antenna Checker with the "Antenna Rules" preferences (in menu **File / Preferences...**, "Tools" section, "Antenna Rules" tab). The dialog lets you modify the required ratio of a layer (poly or metal) to the transistor area.



Arcs in technology: 'mocmos'

Metal-1 (400.0)
Metal-2 (400.0)
Metal-3 (400.0)
Metal-4 (400.0)
Metal-5 (400.0)
Metal-6 (400.0)
Polysilicon-1 (200.0)

Maximum antenna ratio: 400

9−4−1: Introduction to Simulation

Electric has two built−in simulators (described later) and can generate decks for many other simulators. The abilit to interface to external simulators is controlled with the **Simulation (Spice)**, **Simulation (Verilog)**, and **Simulation (Others)** commands (in menu **Tool**).

Be aware that the Electric distribution does not come packaged with these external simulators. You must get your own copy of Spice, Verilog, or any other simulator mentioned here.

Electric can write netlists for these simulators:

Simulator	**Level**	**Netlist Command (in Tool / Simulation)**
ArchSim	functional	**Write ArchSim Deck... (Others)**
CDL	circuit	**Write CDL Deck... (Spice)**
COSMOS	switch	**Write COSMOS Deck... (Others)**
ESIM/RNL	switch	**Write ESIM/RNL Deck... (Others)**
FastHenry	inductance	**Write FastHenry Deck... (Others)**
IRSIM	switch	**Write IRSIM Deck... (Others)**
Maxwell	circuit	**Write Maxwell Deck... (Others)**
MOSSIM	switch	**Write MOSSIM Deck... (Others)**
PAL	gate	**Write PAL Deck... (Others)**
RSIM	switch	**Write RSIM Deck... (Others)**
SILOS	functional	**Write SILOS Deck... (Others)**
Spice	circuit	**Write Spice Deck... (Spice)**
Tegas	functional	**Write Tegas Deck... (Others)**
Verilog	functional	**Write Verilog Deck... (Verilog)**

The ArchSim simulator is a special−purpose experimental simulator. Output from it can be displayed in a waveform window by using **Display ArchSim Journal...** (in menu **Tool / Simulation (Others)**).

For more information on Spice, Verilog, and FastHenry, see the following pages.

9−4−2: Verilog

Electric can produce input decks for Verilog simulation with **Write Verilog Deck...** command (in menu **Tool / Simulation (Verilog)**). After this has been done, you must run Verilog externally to produce a ".dump" file.

Note that the Electric distribution does not come with a Verilog simulator: you must obtain it separately.

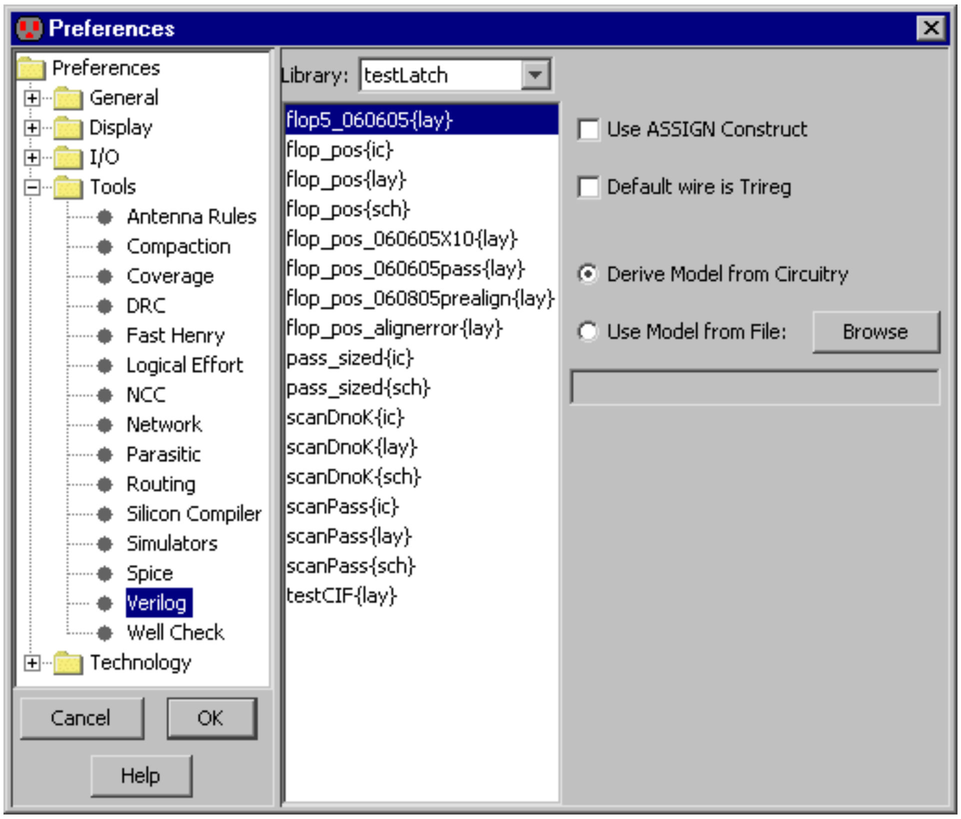
After running a Verilog simulation, you can read the ".dump" file into Electric and display it in a waveform window. This is done with the **Plot Verilog VCD Dump...** command (in menu **Tool / Simulation (Verilog)**). You can also use the **Plot Verilog for This Cell** command if the cell name and file name are the same. The Verilog simulation information is then shown in a digital waveform window (see Section 4−12−1 for more).

Before generating Verilog decks, it is possible to annotate circuits with additional Verilog declarations and code that will be included in the deck. To add Verilog code, select "Verilog Code" under the "Misc." entry in the component menu of the side bar. To add a Verilog declaration, select "Verilog Declaration" under the "Misc." entry in the component menu. These pieces of text can be manipulated like any other text object (see Section 6−8−1 on text).

Additional control of Verilog deck generation is accomplished with the "Verilog" preferences (in menu **File / Preferences...**, "Tools" section, "Verilog" tab). A checkbox lets you choose whether or not to use the Verilog "assign" construct. You can control the type of Verilog declaration that will be used for wires ("wire" by default, "trireg" if checked). Note that this can be overridden with the **Set Verilog Wire** command (in menu **Tool / Simulation (Verilog)**).

Another properety that can be assigned to transistors is their strength. The **Weak** command (in menu **Tool / Simulation (Verilog) / Transistor Strength**) sets the transistor to be weak. The **Normal** command restores the transistor to be normal strength.

The Verilog preferences dialog also lets you attach disk files with Verilog code to any cell in the library. Once attached, the generated Verilog will use the contents of that file instead of examining the cell contents. This allows you to create your own definitions in situations where the derived Verilog would be too complex or otherwise incorrect. For an example of Verilog layout and code, look at the cell "tool−SimulateVERILOG" in the Samples library (get this library with the **Load Library** command, in menu **Help / Samples**).



Library: testLatch

flop5_060605{lay}
flop_pos{ic}
flop_pos{lay}
flop_pos{sch}
flop_pos_060605X10{lay}
flop_pos_060605pass{lay}
flop_pos_060805prealign{lay}
flop_pos_alignerror{lay}
pass_sized{ic}
pass_sized{sch}
scanDnoK{ic}
scanDnoK{lay}
scanDnoK{sch}
scanPass{ic}
scanPass{lay}
scanPass{sch}
testCIF{lay}

☐ Use ASSIGN Construct

☐ Default wire is Trireg

[✓ Derive Model from Circuitry]

Use Model from File:

9−4−3: Spice

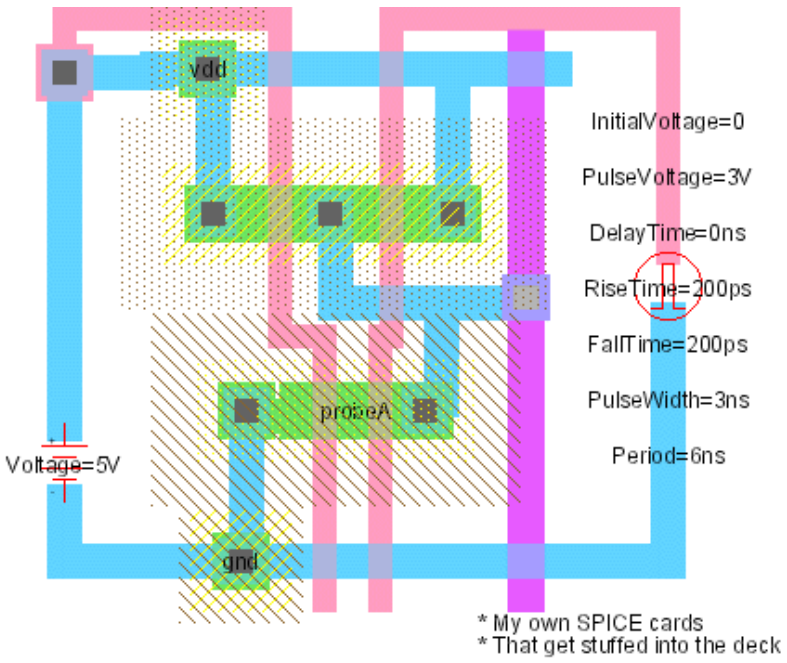
Electric can produce input decks for Spice simulation with **Write Spice Deck...** command (in menu **Tool / Simulation (Spice)**). After this has been done, you must run Spice externally to produce a simulation output file. Note that the Electric distribution does not come with a Spice simulator: you must obtain it separately.

Once Spice has been run, you can see a plot of the simulation by reading the Spice output file back into Electric. Since there are may formats of Spice output, you must first set the "Spice Engine" and the "Output format" fields of the "Spice" Preferences (in menu **File / Preferences...**, "Tools" section, "Spice" tab). The "Output format" field is "Standard" for the default output of the Spice engine; "Raw" for rawfile dumps; and "Raw/Smart" for the rawfile dumps from SmartSpice.

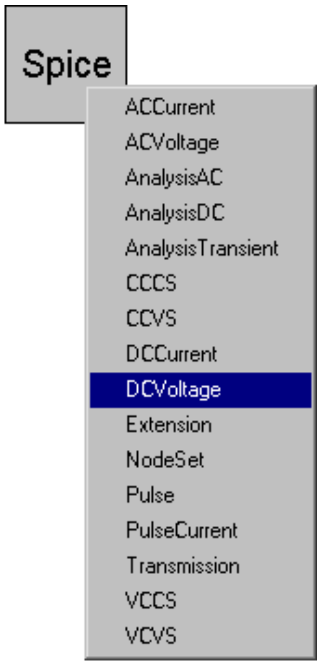
When Electric knows what type of Spice output file to expect, use the **Plot Spice Listing...** command (in menu **Tool / Simulation (Spice)**) to read the file. If the file has the same name as the current cell, you can more simply use **Plot Spice for This Cell**, which does not need to prompt for a file name. The Spice simulation information is shown in an analog waveform window (see Section 4−12−2 for more).

There are many powerful facilities for running Spice with Electric. The example shown here illustrates some of these facilities. This example is available in the Samples library as cell "tool−SimulateSpice" (you can read the library with the **Load Library** command, in menu **Help / Samples**).

All input values to Spice are controlled with special nodes, found in the "Spice" component menu entry. Note that the first time any Spice node is placed, the library of Spice parts is loaded into Electric, so there may be a delay.



The Spice primitives described here are for Electric's default set. However, additional sets can (and have) been written. To choose another set, use the "Spice" preferences (in menu **File / Preferences...**, "Tools" section, "Spice" tab). Under the setting "Spice primitive set", choose another set. A second set, called "SpicePartsS3", is tailored towards special Spice3 primitives.



In this example, there is a 5−volt supply on the left. It was created by using the "DC Voltage" entry under "Spice" entry of the component menu. Once placed, the text that reads "Voltage=0V" can be selected and modified (either with **Object Properties...** or by double−clicking on it). The Pulse input signal on the right is created with the "Pulse" entry under "Spice" (it has 7 parameters).

There are both voltage and current sources, in AC and DC form. The pulse input sources are available as voltage and current. A set of "two−gate" devices are also available: "CCCS", "CCVS", "VCCS", "VCVS", and "Transmission".

It is possible to specify Transient, DC, or AC analysis by using the "Transient Analysis", "DC Analysis", and "AC Analysis" subcommands. Only one such element may exist in a circuit.

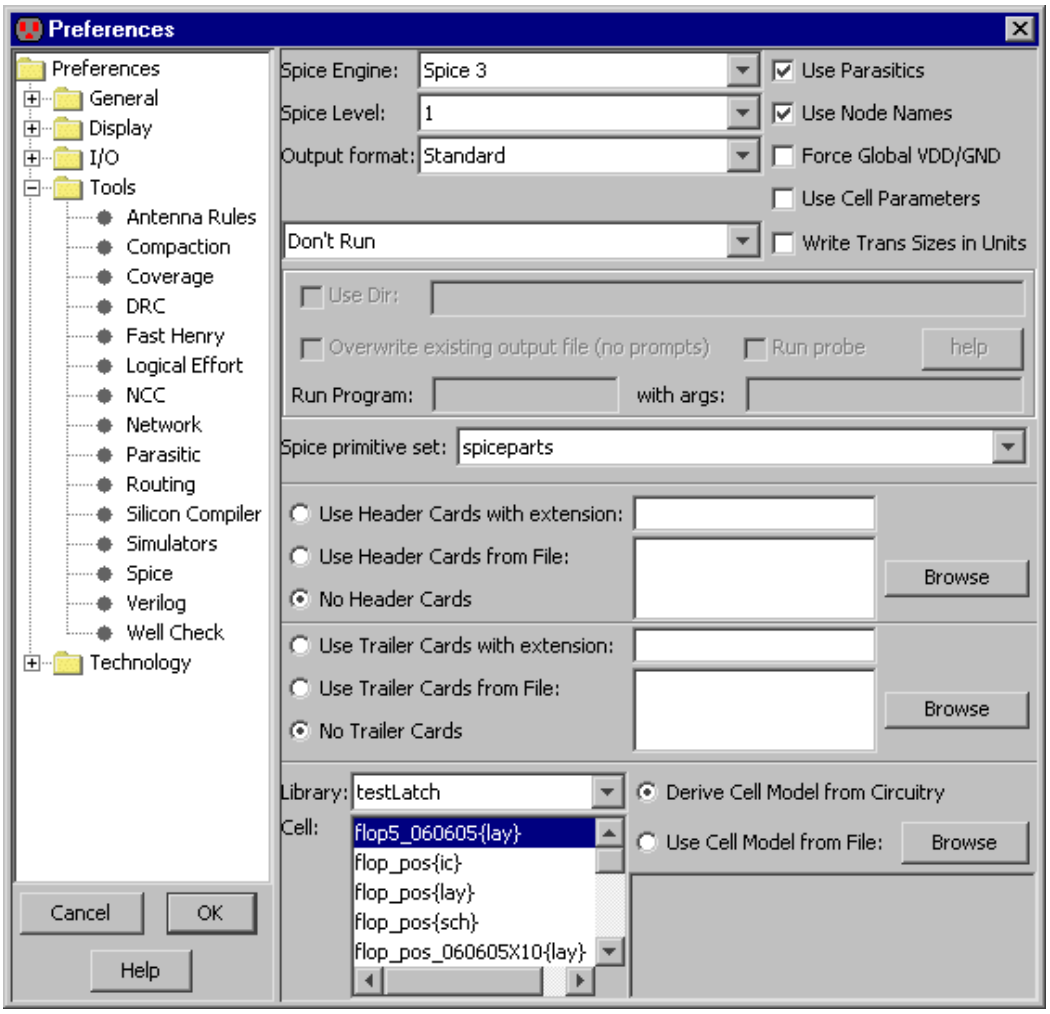
For advanced users, there are two special Spice nodes: "Node Set" and "Extension". The Node Set may be parameterized with an arbitrary piece of Spice code. Truly advanced users may create their own Spice nodes by modifying the cells in the Spice library.

This example also shows the ability to add arbitrary text to the Spice deck, as shown in the lower−right. To create this text, use the "Add Spice Card" under the "Misc." button in the component menu. The command creates a piece of text that can be modified arbitrarily. Whatever the text says will be added to the Spice deck.

Another option that can be used when modeling transistors and other component is to set a specific Spice model to use for that component. To set a node's model, select it and use the **Set Spice Model...** command (in menu **Tool / Simulation (Spice)**).

The **Add Multiplier** subcommand places a multiplier on the currently selected node. Multipliers (also called "M" factors) scale the size of transistors inside of them.

Some nongraphical information can also be given to the Spice simulator with the "Spice" preferences (in menu **File / Preferences...**, "Tools" section, "Spice" tab).



Spice Engine: Spice 3
Spice Level: 1
Output format: Standard
[✓] Use Parasitics
[✓] Use Node Names
[ ] Force Global VDD/GND
[ ] Use Cell Parameters
Don't Run
[ ] Write Trans Sizes in Units
[ ] Use Dir:
[ ] Overwrite existing output file (no prompts)
[ ] Run probe
Run Program:
with args:
Spice primitive set: spiceparts
○ Use Header Cards with extension:
○ Use Header Cards from File:
● No Header Cards
○ Use Trailer Cards with extension:
○ Use Trailer Cards from File:
● No Trailer Cards
Library: testLatch
Cell: flop5_060605{lay}, flop_pos{ic}, flop_pos{lay}, flop_pos{sch}, flop_pos_060605X10{lay}
● Derive Cell Model from Circuitry
○ Use Cell Model from File:

The top part of this dialog allows you to control many of the Spice deck parameters such as:

- **Spice Engine** Can be Spice 2, Spice 3, HSpice, PSpice, Gnucap, or SmartSpice.
- **Spice Level** Can be 1, 2, or 3.
- **Output format** The format to expect when reading Spice output.
- **Use Parasitics** Whether to include simple parasitics in the deck.
- **Use Node Names** Whether to use actual node names in the deck (Spice 2 can only handle numbers).
- **Force Global VDD/GND** Whether to force power and ground to be global signal names.
- **Use Cell Parameters** When set, any parameters defined on the cell will appear in the Spice deck. When not checked, each parameterized cell appears multiple times in the deck, once for each different parameter combination. See Section 6−8−6 for more on parameters.
- **Write Trans Sizes in Units** Requests that the Spice deck contain scalable size information instead of absolute size information.
- **Run Program** Lets Electric run Spice automatically after the deck has been written.
- **Spice Primitive Set** Switches between Spice primitive sets. Currently there are only two: "spiceparts" and "spicepartsG3".

Running Spice: Electric can create an external process as specified by the user to run Spice on the generated netlist. If the pull−down box is set to "Don't Run", nothing is done. If the pull−down box is set to "Run, Ignore Output", the external process is run, and the user is notified when it is finished. If set to "Run, Report Output", a dialog box is opened to show the user the output produced by the process. Please note that this is a *process*, and not a command line command.

There are several options for the process:

- **Use Dir** if specified, this is the working directory of the process.
- **Overwrite existing file** this will overwrite the existing netlist without prompting the user.
- **Run probe** this will run the waveform viewer on the output of the Spice run.
- **Help** tells which environment variables are exported to be used by the process.
- **Run program** the name of the executable to run.
- **with arg** the arguments passed to the executable.

The lower section controls header and trailer cards (placed at the start and end of the Spice deck). This dialog allows you to specify a disk file with cards to be used instead of the built−in set. You can specify a particular file or request that the system search for files with the cell's name and a given extension.

Note that the header and trailer information is specific to a particular technology. If you set this information for one technology, but then use another technology when generating the Spice deck, the information that you set will not be used. Note also that schematics, although a technology in Electric, are not considered to be Spice technology. You can set the proper layout technology that you want to use when dealing with schematics by using the "Use Scale values from this Technology" popup. This popup can be found in the "Technology" preferences (in menu **File / Preferences...**, "Technology" section, "Technology" tab).

The bottom section of the dialog allows you to specify a disk file of Spice cards that will be used to describe any cell. Instead of the cell subcircuit, the specified disk file is "included" in the deck.

9−4−4: Spice and Verilog Primitives

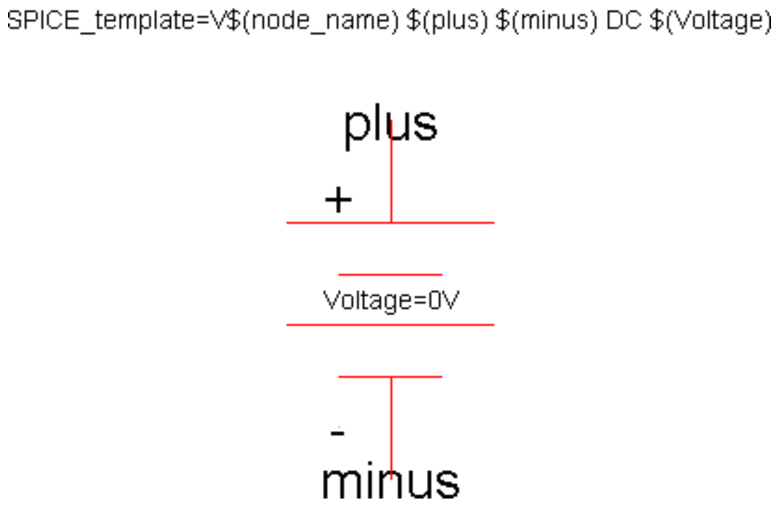
For both Spice and Verilog, you can create special nodes that augment the genetated deck. Spice even has a predefined set of primitives, available from the "Spice" entry in the component menu.

Users can define their own Spice or Verilog elements by creating new icons in this or a new library. The icon should have graphics, exports, optional parameters, and a template. Parameters are created with the **Attribute Properties...** command (in menu **Edit / Attributes**). See Section 6−8−6 for more on parameters. The Spice template is created with the **Set Generic Spice Template** command (in menu **Tool / Simulation (Spice)**). If the template is specific to a particular version of Spice, use the appropriate template command (**Set Spice 2 Template**, **Set Spice 3 Template**, **Set HSpice Template**, **Set PSpice Template**, **Set GnuCap Template**, or **Set SmartSpice Template**).

You can also create Verilog elements by using the **Set Verilog Template** command (in menu **Tool / Simulation (Verilog)**). Note that a single cell can contain both Verilog and multiple Spice templates.

To explain the format of a template, a DC Voltage Source primitive is used as an example. Graphics is placed to describe the look of the symbol (a "battery" look). Exports are created at the top and bottom of the battery with the names "plus" and "minus". A single parameter is defined called "Voltage" with a default value of "0V". Finally, a Spice template is created that has the string

```
 V$(node_name) $(plus)
$(minus) DC $(Voltage)
```



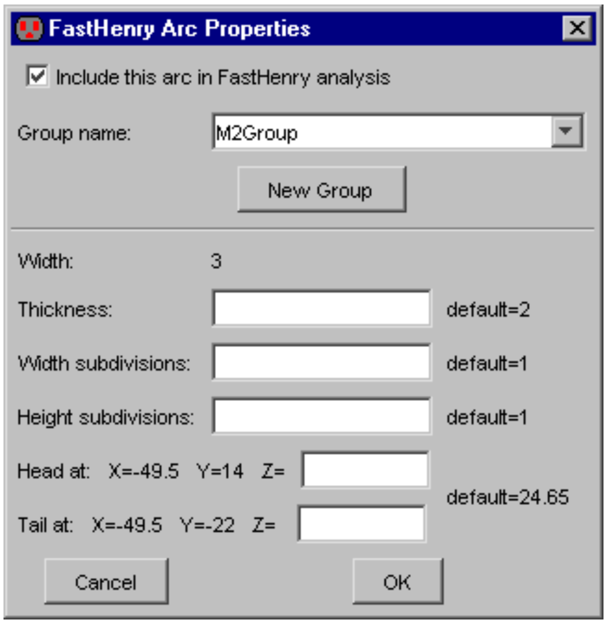
This string contains substitution expressions of the form `$(SOMETHING)` where `SOMETHING` can be an export name, a variable, or "node_name". So, in this example, `$(node_name)` will be replaced with the name of the voltage node; `$(plus)` will be replaced with the net name attached to the positive terminal; `$(minus)` will be replaced with the net name attached to the negative terminal; and `$(Voltage)` will be replaced with the voltage value specified by the user.

The set of Spice primitives in Electric is useful, but far from complete. A second set, called "SpicePartsS3", is tailored towards special Spice3 primitives. There are no Verilog primitives in the current release of Electric. Users who define new primitives are encouraged to share these with the entire community by contacting Static Free Software.

9−4−5: FastHenry

FastHenry is an inductance analysis tool (see the papers of Jacob White). When a FastHenry deck is generated, a subset of the arcs in the current cell are written. To include an arc in the FastHenry deck, select it and use the **FastHenry Arc Properties...** command (in menu **Tools / Simulation (Others)**).

This command presents a dialog with FastHenry factors for the selected arc. The most important factor is at the top: "Include this arc in FastHenry analysis". By checking this, the arc is described in the FastHenry deck. Once this is checked, other fields in the dialog become active. You can set the thickness of this arc (the default value shown will be used if no override is specified). You can set the number of subdivisions that will be used in height and width (again, defaults are shown). You can even set the height of the two ends of the arc.


FastHenry Arc Properties

☑ Include this arc in FastHenry analysis

Group name: M2Group

New Group

Width: 3

Thickness: default=2

Width subdivisions: default=1

Height subdivisions: default=1

Head at: X=-49.5 Y=14 Z=

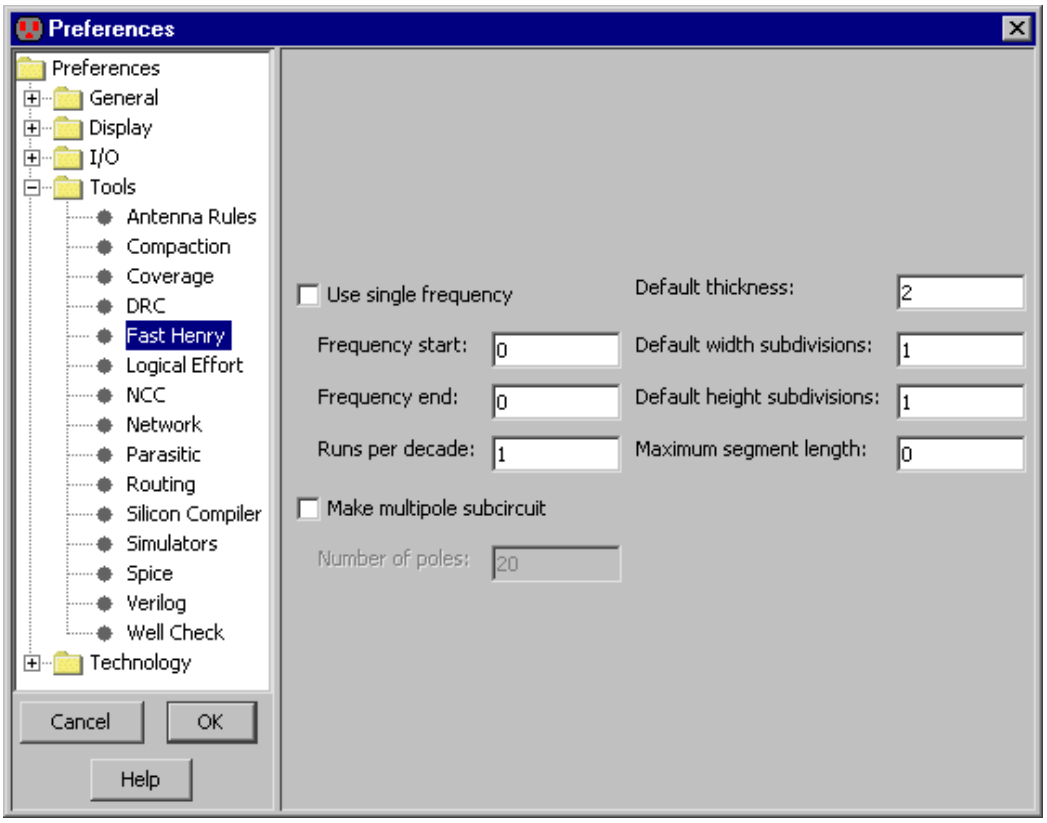
Tail at: X=-49.5 Y=-22 Z=

default=24.65

Cancel OK

You can partition the arcs into different groups. Click the "New Group" button to define a group. After that, arcs can be assigned to one or more groups.

After all arcs have been marked, you can generate a FastHenry deck with the **Write FastHenry Deck...** command (in menu **Tools / Simulation (Others)**). Before doing that, however, you can set other options for FastHenry deck generation. To do this, use the "FastHenry" preferences (in menu **File / Preferences...**, "Tools" section, "FastHenry" tab).



Use single frequency: ☐
Frequency start: 0
Frequency end: 0
Runs per decade: 1
Make multipole subcircuit: ☐
Number of poles: 20
Default thickness: 2
Default width subdivisions: 1
Default height subdivisions: 1
Maximum segment length: 0

This dialog allows you to set the type of frequency analysis (single frequency or a sequence specified by a start, end, and number of runs per decade). You can choose to use single or multiple−pole analysis (and if multiple, you can specify the number of poles). The FastHenry Options dialog also allows you to set defaults for the individual arcs that will be included in the deck. You can specify the default thickness, and the default number of subdivisions (in height and width).

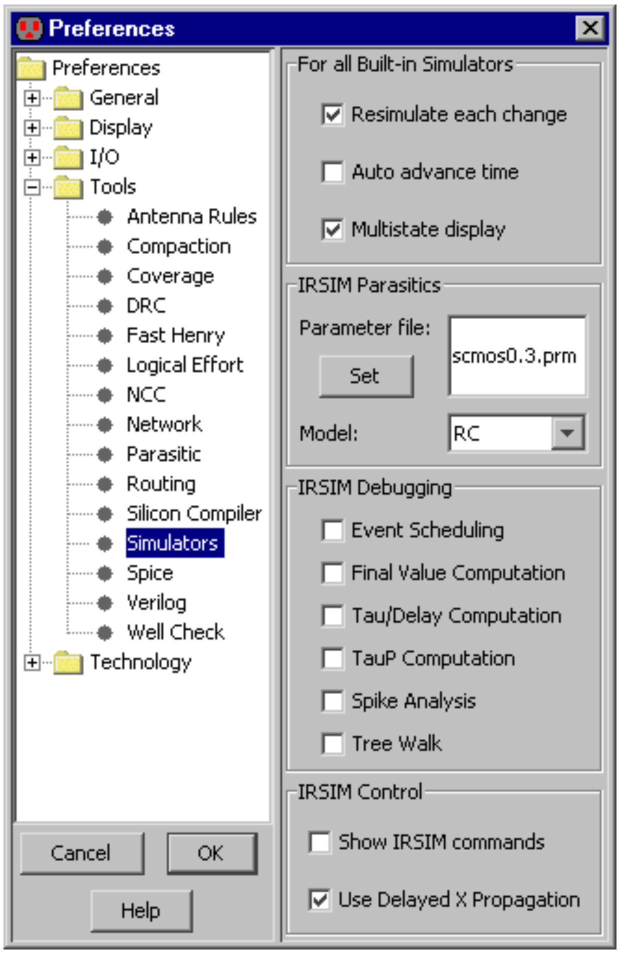
9−5−1: IRSIM

Electric has a built−in simulator, Stanford's IRSIM, which uses RC models to accurately simulate transistors at a gate−level. IRSIM is not packaged with the standard Electric distribution. To obtain it, you must get the additional "plugin" JAR file from Static Free Software (see Section 1−5 for instructions on installing plugins).

To simulate the current cell with IRSIM, use the **IRSIM: Simulate Current Cell** command (in menu **Tool / Simulation (Built−in)**). After issuing this command, a waveform window will appear to control the simulation (see Section 4−12−1 for more). To generate an input deck for IRSIM without running the simulator, use the **IRSIM: Write Deck...** command. To simulate an IRSIM deck (that is, simulate the file, not the circuit), use the **IRSIM: Simulate Deck...** command. Note: if these commands do not appear in the menu, then IRSIM has not been installed.

Since the IRSIM engine is running inside of Electric, you can place stimuli on the circuit and see the results immediately (also described in Section 4−12−1). Note that the command to save stimuli (**Save Stimuli to Disk...** of menu **Tool / Simulation (Built−in)**) writes an IRSIM "command file" which can be edited by hand.

The "Simulators" preferences (in menu **File / Preferences...**, "Tools" section, "Simulators" tab), offers some controls for IRSIM. The general controls at the top are discussed in Section 4−12−1.



For all Built-in Simulators
- [✓] Resimulate each change
- [ ] Auto advance time
- [✓] Multistate display

IRSIM Parasitics
- Parameter file: scmos0.3.prm
- Model: RC

IRSIM Debugging
- [ ] Event Scheduling
- [ ] Final Value Computation
- [ ] Tau/Delay Computation
- [ ] TauP Computation
- [ ] Spike Analysis
- [ ] Tree Walk

IRSIM Control
- [ ] Show IRSIM commands
- [✓] Use Delayed X Propagation

IRSIM uses a parameter file to describe timing and parasitic information. Two of these files come packaged with Electric ("scmos0.3.prm" and "scmos1.0.prm"), but you can create your own and tell IRSIM to use it. In addition to the parameter file, you can select the simulation model that IRSIM uses. The default is a RC model, but a Linear model is also available.

Advanced users who edit their own command files may enter specialized IRSIM debugging commands. These commands depend on a set of flags to determine the type of debugging to do. Checkboxes in the "IRSIM Debugging" section control these debugging flags.

The bottom section has two miscellaneous IRSIM controls.

- "Show IRSIM commands" requests that the system display the command file instructions as they are applied during simulation.
- "Use Delayed X propagation" does less conservative, but potentially more accurate calculation of the time required to propagate an undefined (X) value in the circuit. This improved propagation delay calculation has been shown to be effective in asynchronous circuits.

9−5−2: ALS

Electric has a built−in gate−level simulator called ALS that can simulate schematics, IC layout, or VHDL descriptions. The simulator already knows about MOS transistors and some digital logic gates. It can be augmented with functional descriptions of any circuit using the hardware description language described later in this section.

For an example of ALS simulation, load the "samples" library and simulate the cell "tool−SimulateALS{sch}". You can load the samples library with the **Load Library** command (in menu **Help / Samples**).

To begin simulation of the circuit in the current window, use the **ALS: Simulate This Cell** command (from menu **Tools / Simulation (Built−in)**). After issuing this command, a waveform window will appear to control the simulation (see Section 4−12−1 for more). Since the ALS engine is running inside of Electric, you can place stimuli on the circuit and see the results immediately.

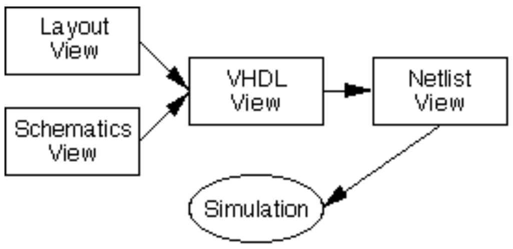
ALS is able to handle transistors with varying strength. To set a transistor to be weak, use the **Weak** command (in menu **Tool / Simulation (Verilog) / Transistor Strength**). To restore the strength to normal, use the **Normal** command. Note that this must be done before simulation begins.

Preferences

The "Simulators" preferences (in menu **File / Preferences...**, "Tools" section, "Simulators" tab) has some controls that affect ALS simulation. The "Multistate display" check tells the simulator to show waveform signals with different colors to indicate different strengths. Without this, a single color is used everywhere. The other general controls at the top are discussed in Section 4−12−1.

9−5−3: ALS Concepts

The user should be aware that the ALS simulator translates the circuit into VHDL, then compiles the VHDL into a netlist for simulation. This means that when a layout or schematic is simulated, two new views of that cell are created: {VHDL} and {net.als}. Use the **Edit VHDL View** (in menu **View**) to see the VHDL code.



When simulation is requested, the cell in the current window is simulated. Date checking is performed to determine whether VHDL translation or netlist compilation is necessary. If you are currently editing a VHDL cell, it will not be regenerated from layout, even if the layout is more recent. Similarly, if you are currently editing a netlist cell, it will not be regenerated from VHDL, even if that VHDL is more recent. Thus, simulation of the currently edited cell is guaranteed.

Note that the presence of VHDL in the path to simulation means that it can simulate VHDL that is entered manually. You can type this VHDL directly into the cell (see Section 4−10 for more on text editing). Also, you can explicitly request that VHDL be produced from schematics or layout with the **Make VHDL View** command (in menu **View**).

This complete VHDL capability, combined with the Silicon Compiler which places and routes from VHDL descriptions, gives Electric a powerful facility for creating, testing, and constructing complex circuits from high−level specifications. See Section 9−12 for more on the Silicon Compiler.

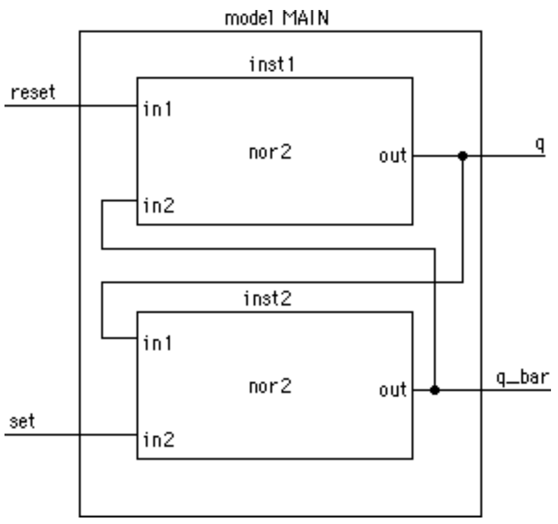
Behavioral Models

When the VHDL for a circuit is compiled into a netlist, both connectivity and behavior are included. This is because the netlist format is hierarchical, and at the bottom of the hierarchy are behavioral primitives. Electric knows the behavioral primitives for MOS transistors, AND, OR, NAND, NOR, Inverter, and XOR gates. Other primitives can be defined by the user, and all of the existing primitives can be redefined.

To create (or redefine) a primitive's behavior, simply create the {net.als} view of the cell with that primitive's name. Use the **New Cell...** command (in menu **Cell**) and select the "netlist.als" view. For example, to define the behavior of an ALU cell, edit "alu{net.als}", and to redefine the behavior of a two−input And gate, edit "and2{net.als}". The compiler copies these textual cells into the netlist description whenever that node is referenced in the VHDL.

The netlist format provides three different types of entities: *model*, *gate*, and *function*. The model entity describes interconnectivity between other entities. It describes the hierarchy and the topology. The gate and function entities are at the primitive level. The gate uses a truth−table and the function makes reference to Java−coded behavior (which must be compiled into Electric, see the module "com.sun.electric.tool.simulation.als.UserCom.java"). Both primitive entities also allow the specification of operational parameters such as switching speed, capacitive loading and propagation delay. (The simulator determines the capacitive load, and thus the event switching delay, of each node of the system by considering the capacitive load of each primitive connected to a node as well as taking into account feedback paths to the node.)

A sample netlist describing an RS latch model is shown below. Note that the "#" character starts a comment.



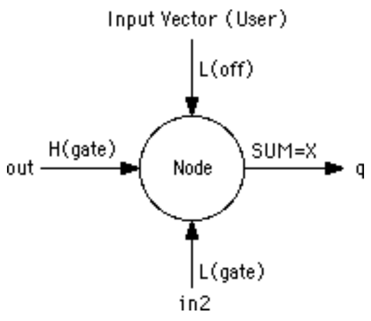
```
# model declaration for the figure
model main(set, reset, q, q_bar)
inst1: nor2(reset, q_bar, q)
inst2: nor2(q, set, q_bar)

# gate description of nor2
gate nor2(in1, in2, out)
t: delta=4.5e−9 + linear=5.0e−10
i: in1=L in2=L   o: out=H@2
i: in1=H         o: out=L@2
i: in2=H         o: out=L@2
i:               o: out=X@2
```

When combined, these entities represent a complete description of the circuit. Note that when a gate, function, or other model is referenced within a model description, there is a one−to−one correspondence between the signal names listed at the calling site and the signal names contained in the header of the called entity.

Simulator Internals

The ALS simulator simulates a set of *simulation nodes*. A simulation node is a connection point which may have one or more *signals* associated with it.

A simulation node can have 3 values (L, H, or X) and can have 4 strengths (off, node, gate, and VDD, in order of increasing strength). It is thus a 12−state simulator. In deciding the state of a simulation node at a particular time of the simulation, the simulator considers the states and strengths of all inputs driving the node.



Driving inputs may be from other simulation nodes, in which case the driving strength is "gate" (i.e. H(gate) indicates a logic HIGH state with gate driving strength), from a power or ground supply ("VDD" strength) or from the user (any strength). If no user vector has been input at the current simulation time, then the input defaults to the "off" strength.

In the above example, the combination of a high and a low driving input at the same strength from the signals "out" and "in2" result in the simulation algorithm assigning the X (undefined) state to the output signal represented by "q". This example also shows the behavior of part of the simulation engine's *arbitration algorithm*, which dictates that an undefined state exists if a simulator node is being driven by signals with the same strength but different states, providing that the strength of the driving signals in conflict is the highest state driving the node.

Another important concept for the user to remember is that the simulator is an *event−driven* simulator. When a simulation node changes state, the simulation engine looks through the netlist for other nodes that could potentially change state. Obviously, only simulation nodes joined by model, gate or function entities can potentially change state. If a state change, or event, is required (based on the definition of the inter−nodal behavior as given by the model, gate or function definition), the event is added to the list of events scheduled to occur later in the simulation. When the event time is reached and the event is fired, the simulator must again search the database for other simulation nodes which may potentially change state. This process continues until it has propagated across all possible nodes and events.

9−5−4: ALS Gates

The gate entity is the primary method of specifying behavior. It uses a truth−table to define the operational characteristics of a logic gate. Many behavioral descriptions need contain only a gate entity to be complete.

The gate entity is headed by the **gate** declaration statement and is followed by a body of information. The gate declaration contains a name and a list of exported simulation nodes (which are referenced in a higher level model description). The format of this statement is shown below:

Format:	gate *name*(*signal1*, *signal2*, *signal3*, ... *signalN*)
Example:	gate nor2(in1, in2, out)
	gate and3(a, b, c, output)

There is no limit on the number of signal names that can be placed in the list. If there is not enough room on a single line to accommodate all the names, simply continue the list on the next line.

The i: and o: Statements (Input and Output)

The **i:** and **o:** statements are used to construct a logical truth table for a gate primitive. The signal names and logical assertions which follow the **i:** statement represent one of many possible input conditions. If the logic states of all the input signals match the conditions specified in the **i:** statement, the simulator will schedule the outputs for updating (as specified in the corresponding **o:** statement). The logical truth table for a two input AND gate is shown below:

```
gate and2(in1, in2, output)
i: in1=H in2=H  o: output=H
i: in1=L        o: output=L
i: in2=L        o: output=L
i:              o: output=X
```

The last line of the truth table represents a default condition in the event that none of the previous conditions are valid (e.g. in1=H and in2=X). It should be noted that the simulator examines the input conditions in the order that they appear in the truth table. If a valid input condition is found, the simulator schedules the corresponding output assignments and terminates the truth table search immediately.

Signal References in the i: Statement

Besides testing the logical values of a signal, the **i:** statement can also compare them numerically. The format of a signal references, which follow the **i:** statement, is show below:

Format:	*signal* <operator> *state_value*
or:	*signal* <operator> *other_signal*
Operators:	= Test if equal
	! Test if not equal
	< Test if less than
	> Test if greater than
Example:	node1 = H

input1 ! input2

node3 < 16

There is no limit on the number of signal tests that can follow an **i:** statement. If there is not enough room on a single line to accommodate all the test conditions, the user can continue the list on the next line of the netlist.

Signal References in the o: Statement

The signal references which follow the **o:** statement are used as registers for mathematical operations. It is possible to set a signal to a logic state and it is possible to perform mathematical operations on its contents. The format for signal references which follow the **o:** statement is shown below:

Format:	*signal* [<operator> *operand* [@ <strength>]]
Operators:	= equate signal to value of operand
	+ increment signal by value of operand
	− decrement signal by value of operand
	* multiply signal by value of operand
	/ divide signal by value of operand
	% modulo signal by value of operand
Strengths:	0 off
	1 node
	2 gate
	3 VDD
Example:	qbar = H@3
	out1 + 3
	out + out1@4
	node1 % modulus_node

It should be noted that the logic state of the operand can be directly specified (such as H, 3) or it can be indirectly addressed through a signal name (such as out1, modulus_node). In the indirect addressing case, the value of the signal specified as the operand is used in the mathematical calculations. The strength declaration is optional and if it is omitted, a default strength of 2 (gate) is assigned to the output signal.

The t: Statement (Time Delay)

The propagation delay time (switching speed) of a gate can be set with the **t:** statement. The format of this statement is shown below:

Format:	t: <mode> = *value* { + <mode> =

	value ... }
Mode:	delta: fixed time delay in seconds
	linear: random time delay with uniform distribution
	random: probability function with values between 0 and 1.0
Example:	t: delta=5.0e−9
	t: delta=1.0e−9 + random=0.2

It is possible to combine multiple timing distributions by using the + operator between timing mode declarations. The timing values quoted in the statement should represent the situation where the gate is driving a single unit load (e.g. a minimum size inverter input).

The **t:** statement sets the timing parameters for each row in the truth table (**i:** and **o:** statement pair) that follows in the gate description. It is possible to set different rise and fall times for a gate by using more than one **t:** statement in the gate description. Assuming that a 2 input NAND gate had timing characteristics of *t(lh)* = 1.0 nanoseconds and *t(hl)* = 3.0 nanoseconds, the gate description for the device would be as follows:

```
gate nand2(in1, in2, output)
t: delta=3.0e−9
i: in1=H in2=H   o: output=L
t: delta=1.0e−9
i: in1=L         o: output=H
i: in2=L         o: output=H
```

This example shows that when both inputs are high, the output will go low after a delay of 3.0 nanoseconds and that if either input is low, the output will go high after a delay of 1.0 nanosecond.

The Delta Timing Distribution of the t: Statement

The Delta timing distribution is used to specify a fixed, non−random delay. The format of a delta timing declaration is shown below:

Format:	delta = *value*
Example:	delta = 1.0
	delta = 2.5e−9

The value associated with the delta declaration represents the fixed time delay in seconds (1.0 = 1 second, 2.5e−9 = 2.5 nanoseconds, etc.)

The Linear Timing Distribution of the t: Statement

The Linear timing distribution is used to specify a random delay period that has a uniform probability distribution. The format of a linear timing declaration is shown below:

Format:	linear = *value*
Example:	linear = 1.0

linear =
2.0e−9

The value associated with the linear declaration represents the average delay time (in seconds) for the uniform distribution. This means that there is an equally likely chance that the delay time will lie anywhere between the bounds of 0 and 2 times the value specified.

The Random Probability Function of the t: Statement

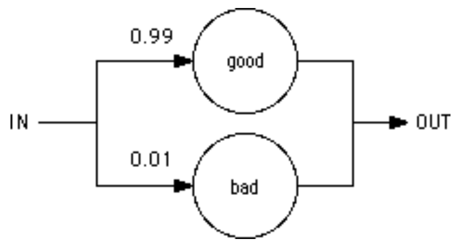
The random probability function enables the user to model things which occur on a percentage basis (e.g. bit error rate, packet routing). The format for random probability declaration is shown below:

Format:	random = *value*
Example:	random = 0.75
	random = 0.25

The value associated with random declaration must be in the range 0.0 <= *value* <= 1.0. This value represents the percentage of the time that the event is intended to occur.

A gate which uses the random probability feature must be operated in parallel with another gate which has a common event driving input. Both these gates should have the same timing distributions associated with them. When the common input changes state, a probability trial is performed. If the probability value is less than or equal to the value specified in the random declaration, the gate containing the random declaration will have its priority temporarily upgraded and its outputs will change state before the outputs of the other gate. This feature gives the user some level of control (on a percentage basis) over which gate will process the input data first.

As an example, a system which models a communication channel that corrupts 1% of the data bytes that pass through it is shown below:



```
model main(in, out)
trans1: good(in, out)
trans2: bad(in, out)
gate good(in, out)
t: delta=1.0e−6
i: in>0x00    o: out=in  in=0x00
gate bad(in, out)
t: delta=1.0e−6 + random=0.01
i: in>0x00    o: out=0xFF in=0x00
```

The netlist describes a system where ASCII characters are represented by 0x01−0x7F. The value 0x00 indicates there is no data in the channel and the value 0xFF indicates a corrupted character. It is assumed that there is an external data source which supplies characters to the channel input. It should be noted that the random declaration is placed on only one of the two gate descriptions rather than both of them. Unpredictable events occur if the random declaration is placed on both gate descriptions.

The Fanout Statement

The **fanout** statement is used to selectively enable/disable fanout calculations for a gate when the database is being compiled. The format for a **fanout** statement is shown below:

Format:	fanout = on
or:	fanout = off

When fanout calculation is enabled (the default setting for all gates), the simulator scans the database and determines the total load that the gate is driving. It then multiplies the gate timing parameters by an amount proportional to the load. If an inverter gate was found to have a propagation delay time of 1 nanosecond when driving a single inverter input, an instance of that gate would have a propagation delay time of 3 nanoseconds if it was driving a load equivalent to 3 inverter inputs.

If fanout calculation is turned off for a gate primitive, fanout calculations for all instances of that gate will be ignored. This feature allows the user to force switching times to a particular value and not have them modified by the simulator at run time.

The Load Statement

The **load** statement is used to set the relative loading (capacitance) for an input or output signal. The format of a **load** statement is shown below:

Format:	load *signal1 = value* { *signal2 = value ...* }
Example:	load in1=2.0 in2=1.5 in3=1.95
	load sa=2.5

The value associated with the signal represents the relative capacitance of the simulation node. When the timing parameters are specified for a gate description, it is assumed that they are chosen for the situation where the gate is driving a single (1.0) unit load such as a minimum size inverter input. The load command tells the simulator that some input structures are smaller or larger (more capacitive) than the reference standard. The simulator, by default, assumes that all signals associated with gate primitives have a load rating of 1.0 (unit load) unless they are overridden by a **load** statement.

The Priority Statement

The **priority** statement is used to establish the scheduling priority for a gate primitive. The format for a **priority** statement is shown below:

Format:	priority = *level*
Example:	priority = 1
	priority = 7

In the event that two gates are scheduled to update their outputs at exactly the same time, the gate with lowest priority level will be processed first. All gate primitives are assigned a default priority of 1 unless they contain random timing declarations in the gate description. In this case the primitive is assigned a default priority of 2. This base priority can be temporarily upgraded to a value of −1 if a random trial is successful during the course of a simulation run. The user is advised to leave the priority settings at their default values unless there is a specific requirement which demands priority readjustment.

The Set Statement

The **set** statement is used to initialize signals to specific logic states before the simulation run takes place. The format for the **set** statement is shown below:

Format:	set *signal1* = <state> @ { <strength> }
	signal2 = <state> @ { <strength> }
Example:	set input1=H@2 input2=L input3=X@0
	set count=4 multiplier=5 divisor=7@2

If the user does not specify a strength value, the signal will be assigned a default logic strength of 3 (VDD). This default setting will override any gate output (because the default strength of 2 is used for gate outputs).

The user will find this feature useful in situations where some of the inputs to a logic gate need to be set to a fixed state for the entire duration of the simulation run. For example, the set and reset inputs of a flip flop should be tied low if these inputs are not being driven by any logic circuitry. All instances of a gate entity which contains a **set** statement will have their corresponding simulation nodes set to the desired state.

9−5−5: ALS Functions

The function entity is an alternate method of specifying behavior. It makes reference to a Java method that has been compiled into Electric. Because there are only a limited number of these methods, and because the source code isn't always easy to update, the function entity is of limited use. However, the facility is very powerful and can be used to efficiently model complex circuits. It permits the designer to work at higher levels of abstraction so that the overall system can be conceived before the low level circuitry is designed. Examples of this include arithmetic logic units, RAM, ROM, and other circuitry which is easier to describe in terms of a software algorithm than a gate level hardware description. To add a function to the simulator, edit the module "com.sun.electric.tool.simulation.als.UserCom.java".

The function entity is headed by a **function** declaration statement that gives a name and a list of exports (which are referenced in a higher level model description). The format of this statement is shown below:

Format:	function *name*(*signal1*, *signal2*, *signal3*, ... *signalN*)
Example:	function JK_FF(ck, j, k, out)
	function DFFLOP(data_in, clk, data_out)
	function BUS_TO_STATE(b7,b6,b5,b4,b3,b2,b1,b0, output)
	function STATE_TO_BUS(input, b7,b6,b5,b4,b3,b2,b1,b0)

The name refers to a Java method, which will find the signal parameters in the same order that they appear in the argument list. The only four functions currently available are listed above. There are two flip−flops (JK and D) and two numeric converters that translate between a bus of 8 signals and a composite hexadecimal digit.

Declaring Input and Output Ports

The **i:** and **o:** statements which follow the function declaration are used to tell the simulator which signals are responsible for driving the function and which drive other events. If any signal in the event driving list changes state, the function is called and the output values are recalculated. The format of an **i:** statement, which contains a list of event driving inputs, is shown below:

Format:	i: *signal1 signal2 signal3 ... signalN*
Example:	i: b7 b6 b5 b4 b3 b2 b1 b0
	i: input phi phi_bar set reset

The format of an **o:** statement which contains a list of output ports is shown below:

Format:	o: *signal1 signal2 signal3 ... signalN*
Example:	o: out1 out2 out3
	o: q q_bar

Other Specifications

Just as there are special statements that affect the operating characteristics of a gate entity, so are these statements available to direct the function entity. The **t:** statement is used to set the time delay between input and output changes. The **load** statement is used to set the relative loading (capacitance) for the input and output ports. The **priority** statement is used to establish the scheduling priority. The **set** statement is used to initialize signals to specific logic states before the simulation run takes place. The format of these statement is identical to that of the gate entity. Note that the Java method does not have to use the values specified in these statements and can schedule events with values that are specified directly inside the code.

Example of Function Use

The specification for a 3 bit shift register (edge triggered) is shown below. This circuit uses a function primitive to model the operation of a D flip−flop:

```
model main(input, ck, q2, q1, q0)
stage0: DFFLOP(input, ck, q0)
stage1: DFFLOP(q0, ck, q1)
stage2: DFFLOP(q1, ck, q2)
function DFFLOP(data_in, clock, output)
i: clock
o: output
t: delta=10e−9
load clock=2.0
```

It should be noted that the clock is the only event driving input for the flip−flop function. There is no need to call the function if the signal "data_in" will be sampled only when the event driving signal ("clock") changes state. The designer can write the function so that it samples the data only when the function is called and the clock input is asserted high (rising edge triggered). If the clock signal is low when the function is called (falling clock edge) the procedure can ignore the data and return control back to the simulation program.

The calling arguments to the Java method are set up as a linked list of signal pointers. The simulator places the arguments into this list in the same order that they appear in the declaration of the function entity. The programmer requires some knowledge of the internals of the simulator to extract the correct information from this list and to schedule new events. A complete discussion of function entity programming is beyond the scope of this document.

9−5−6: ALS Models

As previous examples have shown, the model entity provides connectivity between other entities, including other model entities. The model may be used in conjunction with gate and function entities to describe the behavior of any circuit.

The model entity is headed by a **model** declaration statement and followed by a body which references instances of other entities, lower in the hierarchy. The model name and a list of exports (which are referenced in a higher level model description) are included in this statement. The format of the **model** declaration statement is shown below:

Format:	model *name*(*signal1*, *signal2*, *signal3*, ... *signalN*)
Example:	model dff(d, ck, set, reset, q, q_bar)
	model shift_reg(input, ck, q3, q2, q1, q0)

References to instances of primitive objects (gates and functions) and lower level models are used to describe the topology of the new model to the simulator. The format of an instance reference statement is shown below:

Format:	*instance* : *model* (*signal1*, *signal2*, *signal3*, ... *signalN*)
Example:	gate1: subgate(input, en, mix)
	node5: inverter(mix, out_bar)

It should be noted each instance reference in a model entity must have a unique instance name. The following is an example of the use of a model entity:

```
model latch(input, en, en_bar, out)
gate1: xgate(input, en, mix)
gate2: xgate(out, en_bar, mix)
gate3: inverter(mix, out_bar)
gate4: inverter(out_bar, out)
gate xgate(in, ctl, out)
t: delta=8.0e−9
t: delta=8.0e−9
i: ctl=L       o: out=X@0
i: ctl=H in=L  o: out=L
```

```
   i: ctl=H in=H  o: out=H
   i:             o: out=X@2
   gate inverter(in, out)
   t: delta=5.0e−9
   i: in=L        o: out=H
   i: in=H        o: out=L
   i:             o: out=X@2
```

This example contains the description of a simple latch. When the enable signal is asserted high (en=H, en_bar=L) the input data passes through the transmission gate (gate1) and then through two inverters where it eventually reaches the output. When enable is asserted low (en=L, en_bar=H) the input connection is broken and the feedback transmission gate (gate2) is turned on. The state of the latch is preserved by this feedback path.

The Set Statement

The **set** statement is used to initialize signals within the model description to specific logic states before the simulation run takes place. This feature is useful for tying unused inputs to power(H) or ground(L).

Chapter 9: Tools

9−6−1: Introduction to Routing

The routing tool contains a number of different subsystems for creating wires. Two *stitching* routers can be used in array−based design to connect adjoining cells. A maze−router runs individual wires. A river−router is also available for running multiple parallel wires.

Some of these routers make use of the "Unrouted Arc", a thin−line arc that can connect any two components. Creating "rats nests" of these arcs forms a graphical specification that the router can use. The unrouted arc is from the Generic Technology (see Section 7−6−3). To create one, use the **Get Unrouted Wire** command (in menu **Tool / Routing**). Then use standard wiring commands to run the unrouted arc. Another way to get unrouted wires is to select all or part of an existing route (made with any arc) and use the **Unroute** command.

Finally, the **Copy Routing Topology** and **Paste Routing Topology** commands can be used to create unrouted arcs in one cell (the "pasted" cell) where there are connections of any kind on another cell (the "copied" cell). The **Paste Routing Topology** command uses node and arc names to associate the two cells.

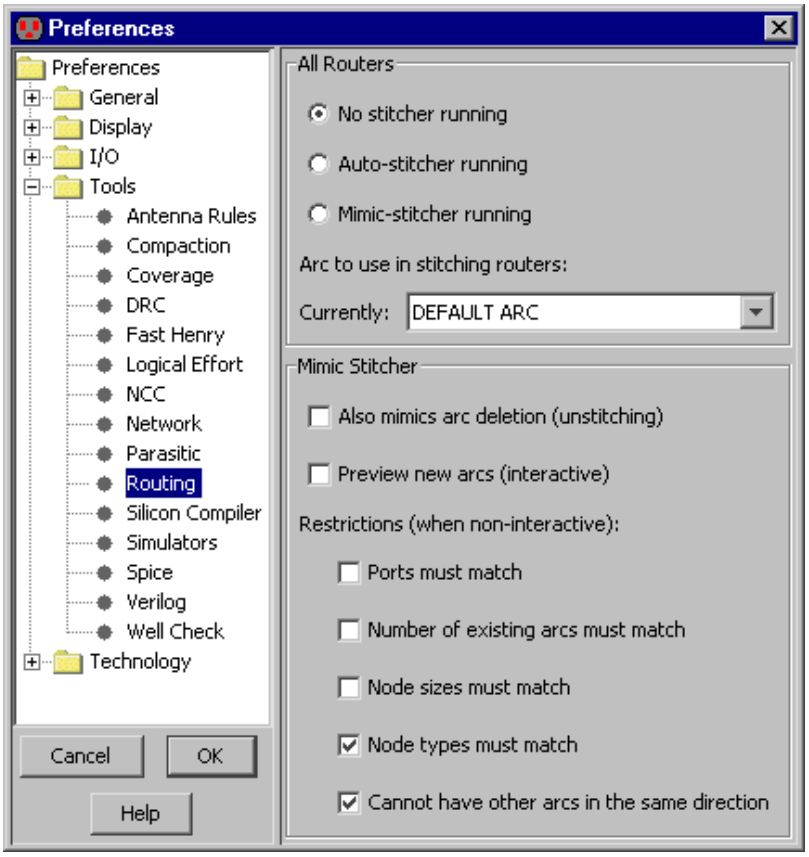
9−6−2: Auto Stitching

The auto−stitching router looks for adjoining nodes that make implicit connections, and places wires at those connections to make them explicit. For example, if a cell has power and ground rails at the top and bottom, and there are ports on the left and right of each rail, then the auto−stitching router can be used to connect all of these rails in a horizontal string of these cells.

The auto−stitcher places a wire when all of these conditions are met:

- The design is layout (auto stitching does not work in schematics).
- Ports exist on both nodes. Because wires must run between two ports, you must make exports at every location where wiring may occur.
- The nodes inside of the cells (the ones with the exports) must touch or overlap, thus creating an implicit connection. When a pin node has an export, it should be the same size as any wires connected to it inside of the cell. This is because a small pin connected to a wide arc will not make an implicit connection when the arc touches something, because the pin is inside of the arc.
- The ports must not already be connected to each other.

To run the auto−stitcher, use the **Enable Auto−Stitching** command (in menu **Tool / Routing**). The router will make all necessary connections, and incrementally add wires as further changes are made to the circuit. To stop stitching, select the menu entry again to uncheck it. To run the auto−stitcher only once for the current cell, use **Auto−Stitch Now** To run it once, and in the highlighted area only, use the **Auto−Stitch Highlighted Now** command. Note that this auto−stitches all cell instances that intersect the highlighted area, so even if only a portion of a cell falls into the highlighted area, the entire cell is stitched.



All Routers

- (●) No stitcher running
- ( ) Auto-stitcher running
- ( ) Mimic-stitcher running

Arc to use in stitching routers:

Currently: DEFAULT ARC

Mimic Stitcher

- [ ] Also mimics arc deletion (unstitching)
- [ ] Preview new arcs (interactive)

Restrictions (when non-interactive):

- [ ] Ports must match
- [ ] Number of existing arcs must match
- [ ] Node sizes must match
- [✓] Node types must match
- [✓] Cannot have other arcs in the same direction

The auto−stitcher allows you to specify a particular type of wire to use in routing. By default, the router figures out which wire to use. However, in the "Routing" preferences (in menu **File / Preferences...**, "Tools" section, "Routing" tab) a specified wire can be given (or automatic selection can be resumed by selecting the "DEFAULT ARC" entry).

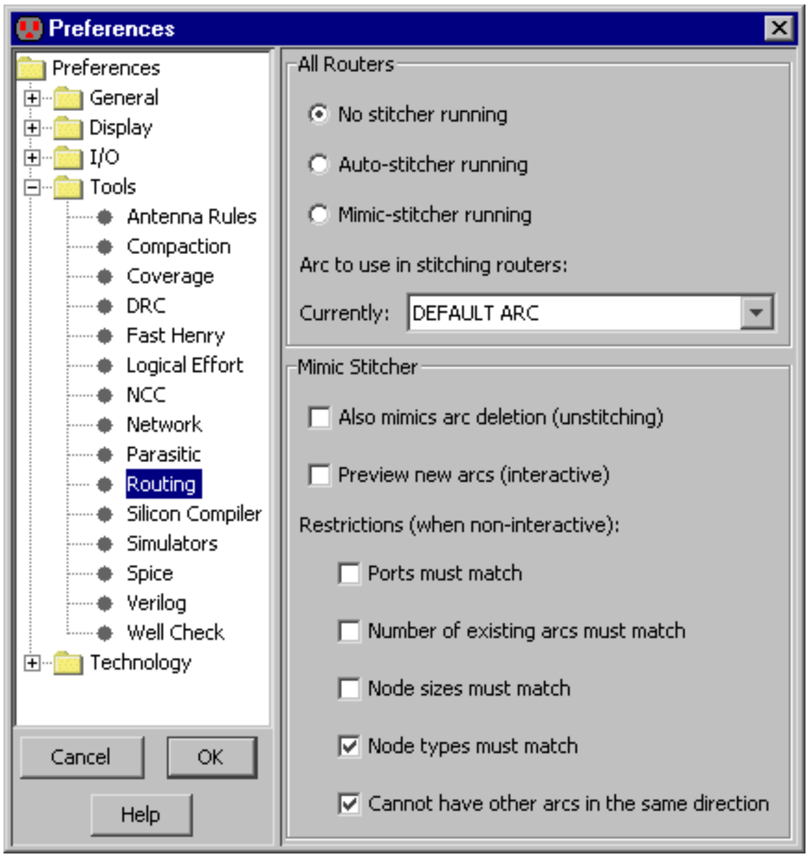
9−6−3: Mimic Stitching

One problem with the auto−stitcher is that it may take a different view of the circuit than originally intended. In an area where more than two cells meet, the auto−stitcher may place many wires in an attempt to connect all touching ports. Another problem with the auto−stitcher is that it makes explicit only what is already implicit, and so does not always add all necessary wires.

To control the wiring of arrays of cells more directly, there is the mimic−stitcher. This tool lets the designer place a wire between two cells, and then it adds other wires between all other similarly configured cells in the circuit. Thus, it mimics your actions.

Specifically, it mimics all situations where the same ports on the same type of nodes exist, separated by the same distance.

The "Routing" preferences (in menu **File / Preferences...**, "Tools" section, "Routing" tab) has many useful controls for mimic stitching.



All Routers

- [✓ No stitcher running]
- Auto-stitcher running
- Mimic-stitcher running

Arc to use in stitching routers:

Currently: DEFAULT ARC

Mimic Stitcher

- ☐ Also mimics arc deletion (unstitching)
- ☐ Preview new arcs (interactive)

Restrictions (when non-interactive):

- ☐ Ports must match
- ☐ Number of existing arcs must match
- ☐ Node sizes must match
- ✓ Node types must match
- ✓ Cannot have other arcs in the same direction

First, you can request that the mimic stitcher also mimic wire deletions. Second, you can request that the mimic stitcher relax its restriction about mimicing arcs (by allowing the ports to be different, the nodes to be different, or the node sizes to be different). You can also ask the mimic stitcher to work interactively, which causes it to examine all possible restriction sets, offering to route wires with increasingly relaxed acceptance criteria.

To turn on the mimic−stitcher, use the **Enable Mimic−Stitching** command (in menu **Tool / Routing**). To disable the stitcher, use the command to uncheck it. You can also request that the mimic−stitcher run just once (mimicing the very last wire that was created or deleted) by using the **Mimic−Stitch Now** command. Finally, you can request that the mimic−stitcher run just once, mimicing the currently selected arc, by using the **Mimic Selected** command.

9−6−4: Maze Routing

The maze router replaces unrouted arcs with actual geometry. To run it, use the **Maze Route** command (in menu **Tool / Routing**). If networks are selected when the command is issued, those networks are routed. If no networks are selected, the all unrouted arcs in the current cell are routed.

Note that maze routing is done one wire at a time, and may fail if no path can be found. Therefore it may be preferable to route the unrouted wires one−at−a−time in order to better control the process.

Note also that maze routing constructs an array which is the size of the route, and searches the array for a routing path. Therefore, long wires will use large amounts of memory and time.

For an example of maze routing, open the Samples library and edit the cell "tool−RoutingMaze" (you can read the library with the **Load Library** command, in menu **Help / Samples**). This cell has a number of unrouted wires that can be routed.

9−6−5: River Routing

River routing is the running of multiple parallel wires between two parallel rows (presumably along facing sides of two cell instances). The wires must remain in sequential order and cannot cross each other. Thus, they appear as a flowing stream of lines, and have the appearance of a river.

To specify an intended path for the river−router, every connection must be made with an Unrouted arc. Thus, before river−routing, there should be a series of direct (and presumably nonmanhattan) unrouted arcs. These arcs are replaced with the appropriate geometry during river−routing.

To convert the unrouted wires into layout, use the **River−Route** command (in menu **Tool / Routing**). If there are unrouted arcs selected, these will be the only ones converted. Otherwise, all unrouted arcs in the cell will be converted. If it is necessary, nodes may be moved to make room for the river−routed wires.

The river router always routes to the left or bottom side of the routing channel. Thus, if there is a vertical channel that is very wide, the wires will run to the left side and then jog to their proper location there. The only way to force routing to the right or top side is to rotate the entire circuit so that these sides are on the left and bottom.

For an example of river routing, open the Samples library and edit the cell "tool−RoutingRiver" (you can read the library with the **Load Library** command, in menu **Help / Samples**).

9−7−1: NCC Overview

Improvements

Electric can compare two different cells and determine whether their networks have the same topology. This operation is sometimes called Layout vs. Schematic (LVS), but because Electric can compare any two circuits (including two layouts or two schematics) we use the term Network Consistency Checking (NCC).

The Java Electric NCC differs from the C Electric NCC in two signficant ways.

- The Java Electric NCC firsts attempts to discover circuit mismatches using a new algorithm called "Local Paritioning". We do this because Local Partitioning provides much more precise and intelligible mismatch diagnostics. We fall back upon the Gemini algorithm (Ebeling, Carl, "GeminiII: A Second Generation Layout Validation Program", *Proceedings of ICCAD* 1988, p322−325.) only as a last resort. In practice upwards of 95% of all errors are found by Local Partitioning.
- The Java Electric NCC has a "hierarchical" mode. When comparing a cell hierarchically, NCC first tries to compare the cell's descendents. We *strongly* recommend this mode to the user because it allows the Local Partitioning algorithm to provide even more precise and intelligible mismatch diagnostics.

The Java Electric NCC is also significantly faster than the C Electric NCC. For example, for one of our chips, C−NCC flat took 48 minutes, Java−NCC flat took 3.5 minutes, and Java−NCC hierarchical took 15 seconds.

Limitations

NCC has a number of limitations

- NCC does not check the substrate connection of transistors. This is because neither C− nor Java−Electric keep track of the connectivity of the substrate connection of layout transistors. In fact, layout transistors don't have substrate ports. We plan to remedy this.
- NCC treats all resistors as short circuits. This is the desired behavior for most of the resistors produced by the Asynchronous Design Group at Sun. However, it would be nice if certain resistors in schematics could be matched up against layout resistors. However, neither C− nor Java− Electric support explicit layout resistors. We plan to remedy this.
- NCC's error messages are completely text based. There is no integration into the GUI. We plan to remedy this.

Example

For an example of network consistency checking, open the Samples library (in menu **Help / Samples / Load Library**) and compare the cells "tool−NCC{lay}" and "tool−NCC{sch}". These two cells are equivalent and the checker will find them to be so.

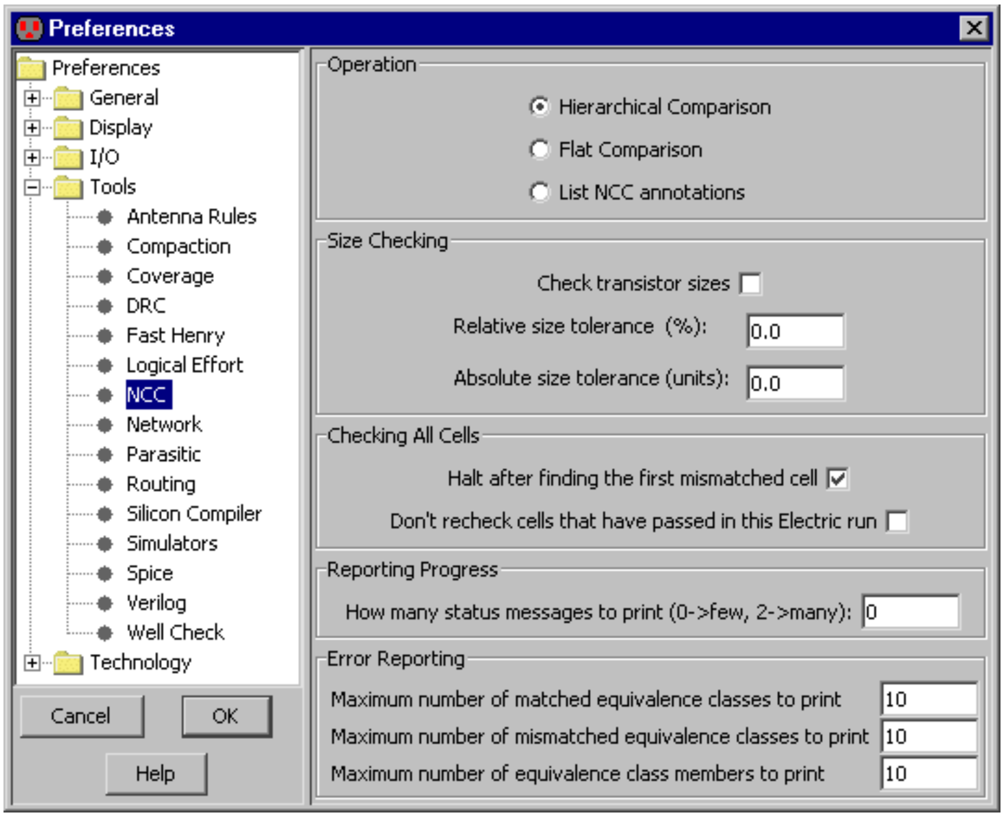
9−7−2: NCC Commands

To compare two cells, use these commands (in menu **Tool / NCC**):

- **Schematic and Layout Views of Cell in Current Window** Use a heuristic to figure out what to compare against the cell in the current window. If the current cell is a schematic then compare it against some layout cell in the same cell group. If the current cell is a layout then compare it against some schematic cell in the same cell group. Since most cell groups have one layout cell and one schematic cell, this form of the NCC command is usually the most convenient.
- **Cells from Two Windows** Compare the two cells that are displayed in the two opened windows (there must be exactly two windows). This is useful when the schematic and layout are not in the same cell group. The command can also be used to compare schematics with schematics or layout with layout. However, the command refuses to compare icon cells since icons cells don't have defined connectivity.

9−7−3: NCC Preferences

NCC options are available in the "NCC" preferences (in menu **File / Preferences...** , "Tools" section, "NCC" tab).



Operation
[✓ Hierarchical Comparison]
Flat Comparison
List NCC annotations

Size Checking
Check transistor sizes ☐
Relative size tolerance (%): 0.0
Absolute size tolerance (units): 0.0

Checking All Cells
Halt after finding the first mismatched cell ✓
Don't recheck cells that have passed in this Electric run ☐

Reporting Progress
How many status messages to print (0->few, 2->many): 0

Error Reporting
Maximum number of matched equivalence classes to print 10
Maximum number of mismatched equivalence classes to print 10
Maximum number of equivalence class members to print 10

Operation Section

The "Operation" section allows you to select what kind of NCC operation to perform. You can either compare hierarchically, compare flat, or list all the NCC annotations in the design.

We recommend hierarchical over flat comparison because hierarchical comparisons are faster and the mismatch diagnostics are much more precise and intelligible. However, transistor size checking of schematics with automatically sized transistors does not work with hierarchical comparisons. If you select "Check transistor sizes" and your schematics include automatically sized transistors you must also select "Flat Comparison".

The best way to use NCC is to initially perform all your comparisons hierarchically. This will typically require many iterations. Once you have gotten your designs to pass hierarchical comparison, turn on size checking and flat comparison. This will report transistor size mismatches.

Size Checking Section

The "Size Checking" section controls the checking of transistor widths and lengths.

NCC does the following when size checking is enabled. After each topological comparison, NCC checks if it found any topological mismatches. If NCC found no mismatches then NCC checks, for each pair of matching transistors, that the widths and lengths are approximately equal.

The two tolerance boxes allow the user to specify how much more the larger of the two matched transistors may be than the smaller before NCC reports a size mismatch. The "Relative size tolerance" box specifies the difference in percentage. The "Absolute size tolerance" box specifies the difference in units. NCC reports a size mismatch when both tolerances are exceeded.

If you enable "Size Checking" and "Hierarchical Comparison" simultaneously then NCC restricts which Cells it treats hierarchically to ensure a correct answer in the presence of automatically sized transistors. For this case it compares a pair of Cells hierarchically if and only if each Cell is instantiated exactly once.

Checking All Cells Section

In hierarchical mode NCC attempts to compare all cells in the design starting with those at the leaves and working it's way toward the root. For that mode it is often best if NCC stops as soon as it finds a mismatch. To get this behavior the user should check the box: "Halt after finding the first mismatched cell".

However, it is occasionally useful to continue checking even after mismatches have been detected. For example, the designer might find that although cell ABC mismatches, she is unable to fix ABC because someone else designed it. When asked to continue, NCC will do the following when comparing cells that use ABC:

- If NCC found no export mismatches when comparing ABC then NCC will use the export names to identify corresponding ports in the layout and schematic.
- If NCC found export mismatches when comparing ABC then NCC will flatten the one level of hierarchy: ABC, before performing the comparison.

The check box "Don't recheck cells that have passed in this Electric run" skips the checking of a pair of cells if they have ever passed in this run of Electric. Because this command *is not* smart enough to recheck the cells after either have changed, this command is of very limited utility. At the moment NCC has run sufficiently fast that it doesn't seem worth the effort to implement anything more sophisticated.

Reporting Progress Section

This panel controls how verbose NCC is in reporting its progress. Most users should leave this at 0.

Error Reporting Section

The error reporting section controls how many error messages are printed when the Local Partitioning algorithm has failed to find a mismatch but the Gemini algorithm has. Most users will want to leave these at the default setting of 10.

9−7−4: NCC Annotations

For certain situations NCC cannot figure out that two cells are equivalent unless the designer supplies extra information. The designer supplies this extra information by adding NCC annotations to layout and/or schematic cells. NCC annotation's are represented by attributes placed on cells (see Section 6−8−5). The attribute's name is *NCC*. The attribute's value is one or more lines. Each line contains a separate NCC annotation. Thus, although a Cell can have at most one attribute named *NCC*, that attribute can contain any number of NCC annotations.

exportsConnectedByParent *<string or regular expression>*

Layout cells sometimes contain multiple exports that are supposed to be connected by the parent cell. For example, a layout cell, A{lay}, might export vdd, vdd_1, vdd_2, and vdd3. The designer expects the cell that instantiates A{lay} will connect all the vdd exports to a single net: vdd. However, because the corresponding schematic cell usually only contains a single export, vdd, the NCC of the schematic and layout cells fails. This situation is most common for the power and ground networks, although it occasionally arises for signal networks such as *clock* or *precharge*.

NCC allows the designer to add the annotation: exportsConnectedByParent to the cell to inform NCC which exports will be connected by the parent. The keyword is followed by a list of strings and/or regular expressions. A string matches an export name exactly, for example: vdd. Thus A{lay} can contain the NCC annotation:

exportsConnectedByParent vdd vdd_1 vdd_2

Alternatively, the designer can use regular expressions. Regular expressions begin and end with the character: '/'. Thus A{lay} can contain the NCC annotation:

exportsConnectedByParent vdd /vdd_[0−9]+/

When NCC compares a cell with an *exportsConnectedByParent* annotation it performs the comparison as if those exports were connected. It is safe for NCC to believe this annotation because NCC also checks the assertion. When NCC encounters an instance of a cell with an *exportsConnectedByParent* annotation NCC reports an error whenever that assertion isn't satisfied.

skipNCC *<comment>*

The skipNCC annotation should be added to a cell, say B, when:

- B{sch} and B{lay} won't pass either flat or hierarchical NCC and
- you want any hierarchical NCC of the parents of B to flatten the one level of hierarchy: cell B.

If cell B has a skipNCC annotation, then a hierarchical comparison won't check B and will simply flatten through the one level of hierarchy: B.

All the characters following the keyword to the end of the line serve as a comment. This is useful for documenting why this annotation was necessary. When you ask NCC to compare every cell in the design, NCC will tell you which cells it is skipping and why. For example, if cell B includes the NCC annotation:

skipNCC layout is missing ground connection

then NCC will print:

Skipping NCC of A because layout is missing ground connection.

A common reason for needing this annotation is an unfortunate situation: the exports of B{sch} and B{lay} don't match. A skipNCC on B prevents NCC from reporting export mismatches because 1) cell B is not checked by itself and 2) when a parent of cell B is checked, B's exports are discarded when NCC flattens through cell B. Although not always possible, it's better to fix export mismatches, because fixing them will yield clearer mismatch diagnostics when there is a problem.

flattenInstances <*string or regular expression*> ...

Hierarchical NCCs do not require a perfect match between the schematic and layout hierarchies. Instead, hierarchical NCC uses heuristics to determine which cell instances must be flattened and which can be compared hierarchically. The heuristic sometimes make mistakes. When that happens, the flattenInstances annotation can guide the heuristic.

The list of strings and/or regular expressions are used to match instance names within the cell. Those cell instances that match are always flattened.

notSubcircuit <*comment*>

The designer should add the notSubcircuit annotation to a cell, say B, if:

- B{sch} and B{lay} will pass NCC when compared separately but
- hierarchical NCC of a parent of B should not treat B as a hierarchical element but should, instead, flatten through B.

One reason for using this annotation is to correct errors made by the heuristic that determines which cells to flatten and which to compare hierarchically. For example, suppose that the schematics instantiate cell B{sch} 1000 times and the layout instantiates cell B{lay} 500 times. In principle one could use the *flattenInstances* annotation to inform NCC which instances to keep and which to flatten. However sometimes that's more work than it's worth and it's better to add a single *notSubcircuit* annotation to cell B{sch} or B{lay} to tell NCC to never treat B as a hierarchical entity.

When hierarchical NCC encounters a notSubcircuit annotation it prints a message that includes the comment in a manner similar to skipNCC.

The notSubcircuit annotation only affects hierarchical NCCs, it is ignored by flat NCCs.

joinGroup <*cell name*>

The designer should add a joinGroup annotation to, say, cell B if NCC should behave as if cell B belonged to a different cell group and that cell group is in a different library. The cell group to move B to is that cell group that contains <*cell name*>. That specification should be fully qualified: library:cell{view}.

Memberships in cell groups is important when NCC performs hierarchical comparisons because NCC assumes that cells in the same cell group are supposed to be topologically identical. Membership of two cells in the same cell group is one criteria NCC uses to decide that it should treat them as hierarchical entities and it should compare them separately.

Occasionally it is impractical to place the layout and schematic views of a cell in the same cell group. For example when layout is automatically generated from hand drawn schematics it may be better to place the layout in a different library than the schematics.

blackBox <*comment*>

Don't compare the Cells in this Cell group; just assume they are topologically equivalent.

The *blackBox* is useful when a particular arrangement of layout geometry implemements a construct that Electric doesn't understand. For example, we have used this construct to handle resistors and parasitic bipolar transistors in the layout.

9−8−1: Pad Frame Generation

The Pad Frame generator reads a disk file and places a ring of pads around your chip. The pads are contained in a separate library, and are copied into the current library to construct the pad frame.

The format of the pad frame disk file is as follows:

```
celllibrary LIBRARYFILE [copy]          ; Identifies the file with the pads
cell PADFRAMECELL                       ; Creates a cell to hold the pad frame
core CORECELL                           ; Places cell in center of pad frame
align PADCELL INPUTPORT OUTPUTPORT      ; Defines input and output ports on pads
export PADCELL IOPORT [COREPORT]        ; Defines exports on the pads
place PADCELL [GAP] [PORTASSOCIATION]   ; Places a pad into the pad frame
rotate DIRECTION                        ; Turns the corner in pad placement
```

The file must have exactly one `celllibrary` and `cell` statement, as they identify the pad library and the pad frame cell. If the `celllibrary` line ends with the keyword `copy`, then cells from that library are copied into the library with the pad ring (by default, they are merely instantiated, creating a cross−library reference to the pads library). The file may have only one `core` statement to place your top−level circuit inside of the pad frame. If there is no `core` statement, then pads are placed without any circuit in the middle.

The `align` statement is used to identify connection points on the pads that will be used for placement. Each pad should have an input and an output port that define the edges of the pad. These ports are typically the on the power or ground rails that run through the pad. When placing pads, the output port of one pad is aligned with the input port of the next pad.

Each pad that is placed with a `place` statement is aligned with the previous pad according to the alignment factor. A `gap` can be given in the placement that spreads the two pads by the specified distance. For example, the statement:

```
place padIn gap=100
```

requests that the "padIn" pad be placed so that its input port is 100 units away from the previous pad's output port.

If a core cell has been given, you can also indicate wiring between the pads and the core ports. This is done by having one or more *port associations* in the `place` statements. The format of a port association is simply `PADPORT = COREPORT`. For example, the statement:

```
place padOut tap=y
```

indicates that the "tap" port on the placed pad will connect to the "y" port on the core cell.

The port association can also create an export on the pad. The statement:

```
place padOut export io=o7 export tap=core_o7
```

creates two exports on the pad, "o7" on its "io" port, and "core_o7" on its tap port. For many instances of this pad type, this notation can be condensed with the use of the `name` keyword in conjunction with exports defined for the pad at the start of the file. For example, defining the IO ports as

```
export padOut io tap
```

and then changing the place statement to

```
place padOut name=o7
```

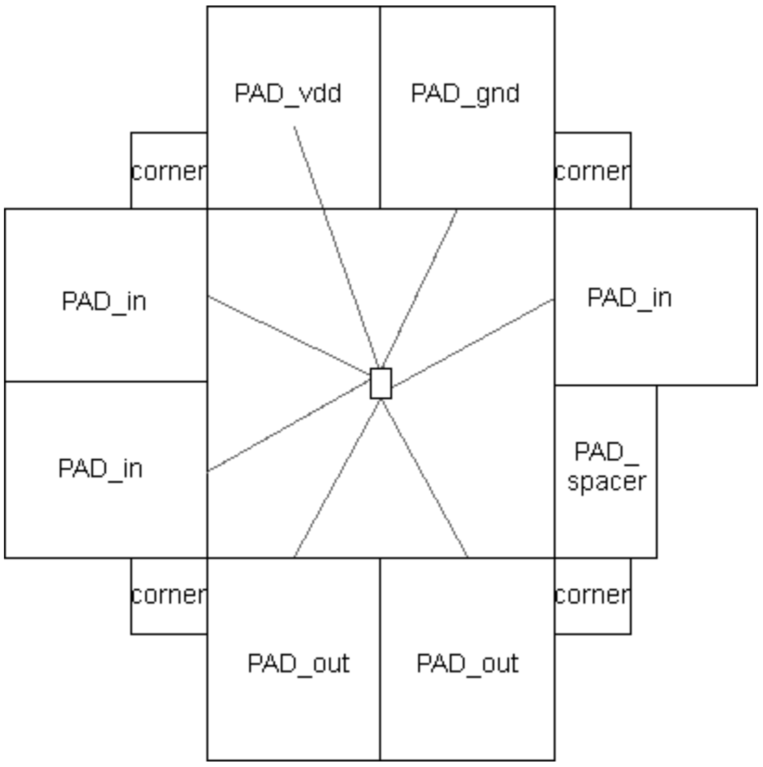
results in the same ports being exported with the same names. This shorted notation always prepends name with "core_" on the core port export.

The `rotate` statement rotates subsequent pads by the specified amount. The statement has only two forms: `rotate c` to rotate clockwise, and `rotate cc` to rotate counterclockwise.

Here is an example of a pad frame disk file, with the finished layout. There is a cell in the Samples library called "tool−PadFrame" (get it with the **Load Library** command, in menu **Help / Samples**). Suppose a text file is created with with this content, and read with the **Pad Frame Generator...** command (in menu **Tool / Generation**).

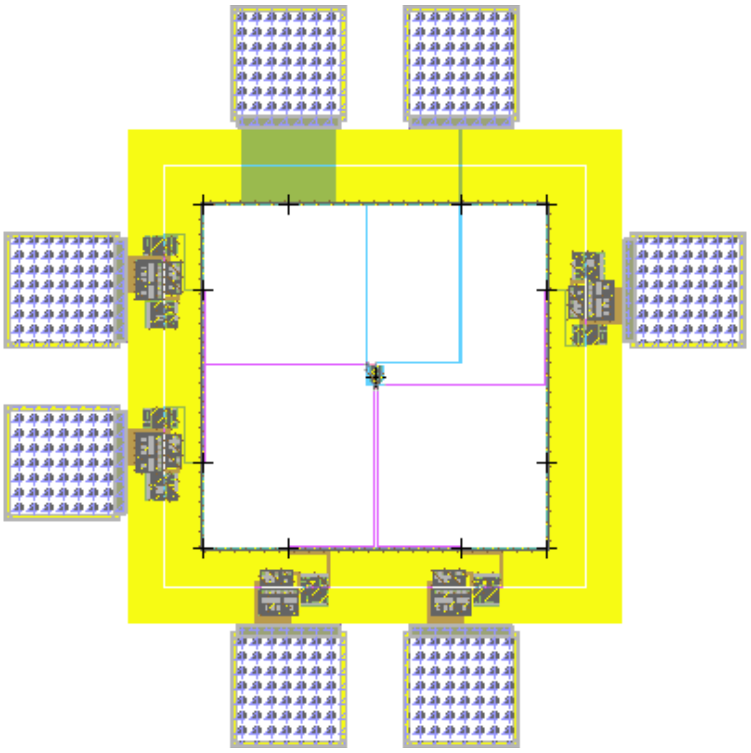
```
; specify library with pads
celllibrary pads4u.txt

; create a cell called "padframe"
cell padframe

; place this cell as the "core"
core tool−PadFrame

; set the alignment of the pads
;  (with input and output export)
align PAD_in{lay}  dvddL dvddR
align PAD_out{lay}  dvddL dvddR
align PAD_vdd{lay}  dvddL dvddR
align PAD_gnd{lay}  dvddL dvddR
align PAD_corner{lay} dvddL dvddR
align PAD_spacer{lay} dvddL dvddR
```

```
; place the top edge of pads
place PAD_corner{lay}
place PAD_gnd{lay} gnd_in=gnd
place PAD_vdd{lay} m1m2=vdd

; place the right edge of pads
rotate c
place PAD_corner{lay}
place PAD_in{lay} out=pulse
place PAD_spacer{lay}

; place the bottom edge of pads
rotate c
place PAD_corner{lay}
place PAD_out{lay} in=out1
place PAD_out{lay} in=out2

; place the left edge of pads
rotate c
place PAD_corner{lay}
place PAD_in{lay} out=in1
place PAD_in{lay} out=in2
```



This file places 8 pads in a ring (2 on each side) and also places corner "pads" for making bends. The input pads connect to the 2 input ports "a1" and "a2". The output pads connect to the 3 output ports "out1", "out2", and "out3" The power and ground pads connect to the "vdd" and "gnd" ports.

Note that the generator places pad instances, but does not wire them to each other. In order to create a uniform ring of power and ground between the pads, you can use the Auto−router or the Mimic−router (see Section 9−6−1).



Connections between pads and ports of the core cell use Unrouted arcs (from the Generic technology, see Section 7−6−3). After these connections are routed with real geometry, the finished layout is shown here, fully instantiated.

9−8−2: Other Generators

There are other generators built into Electric. These commands (in menu **Tool / Generation**) may be used:

- **Coverage Implants Generator** Although individual MOS nodes and arcs have the proper amount of implant around them, a collection of such objects may result in an irregular implant boundary. To clean this up, you can place pure−layer nodes of implant that neatly cover the implant area. This command does it automatically. It removes previous pieces of coverage implant before running, so that the result is a clean cover.
- **ROM Generator...** The ROM generator constructs many cells to describe a ROM from a personality file. You will be prompted for the personality file. The first line of the ROM personality file lists the degree of folding. For example, a 256−word x 10−bit ROM with a folding degree of 4 will be implemented as a 64 x 40 array with 4:1 column multiplexers to return 10 bits of data while occupying more of a square form factor. The number of words and degree of folding should be powers of 2. The remaining lines of the file list the contents of each word. The parser is pretty picky. There should be a carriage return after the list word, but no other blank lines in the file. Here is a sample ROM file:
```
1
010101
011001
100101
101010
4
00000000
10000000
01000000
11000000
```
- **MOSIS CMOS PLA Generator...** The MOSIS CMOS PLA generator reads two personality files (AND and OR) and generates a PLA array. Each file has only two numbers on the first line to define the size of the array, and the values of the array on subsequent lines. Both the AND file and the OR file are similar. Example files can be found in the **PLA−ROM** subdirectory of the **examples** directory. Here is some sample PLA logic:

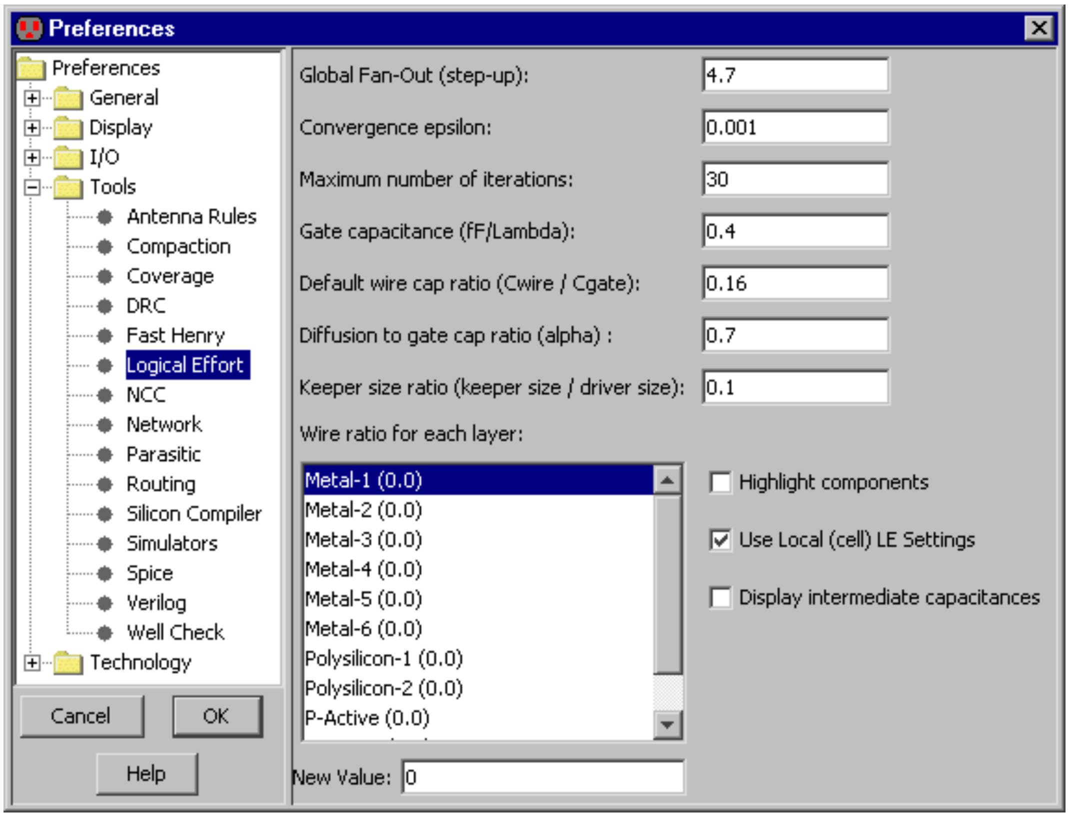
  $$f = (a \textbf{ and } b \textbf{ and } (\textbf{not } c)) \textbf{ or } ((\textbf{not } b) \textbf{ and } (\textbf{not } a))$$
  $$g = (a \textbf{ and } c) \textbf{ or } ((\textbf{not } a) \textbf{ and } (\textbf{not } c))$$

  Here is the AND file for the above logic:
```
4  3
1  1  0
0  0  X
1  X  1
0  X  0
```
- **Generate gate layouts (MoCMOS)** Generates a set of gates in the MOSIS CMOS technology.

9−9: Logical Effort

The Logical Effort tool examines a digital schematic and determines the optimal transistor size to use in order to get maximum speed. The tool is based on the book *Logical Effort*, by Ivan Sutherland, Bob Sproull, and David Harris (Morgan Kaufmann, San Francisco, 1999). It is highly recommended that the user be familiar with the concept of this book before using the Logical Effort Tool.

To control Logical Effort, use the "Logical Effort" preferences (in menu **File / Preferences...**, "Tools" section, "Logical Effort" tab). This lets you control a number of settings for Logical Effort analysis.

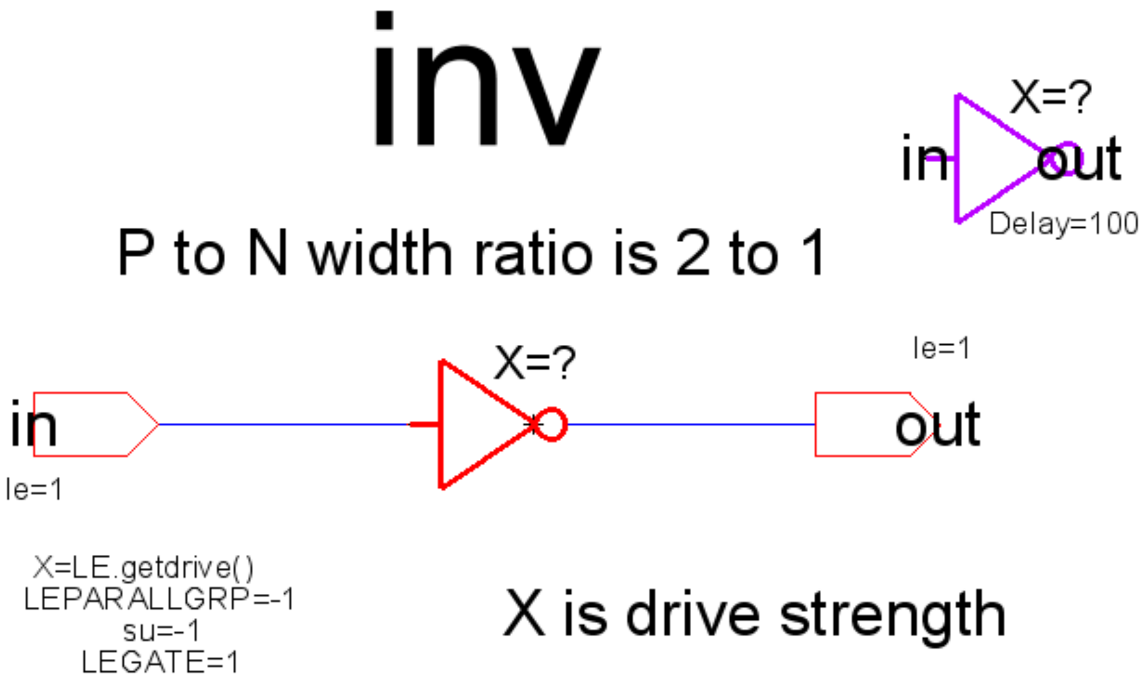


Global Fan-Out (step-up): 4.7
Convergence epsilon: 0.001
Maximum number of iterations: 30
Gate capacitance (fF/Lambda): 0.4
Default wire cap ratio (Cwire / Cgate): 0.16
Diffusion to gate cap ratio (alpha) : 0.7
Keeper size ratio (keeper size / driver size): 0.1
Wire ratio for each layer:
Metal-1 (0.0)
Metal-2 (0.0)
Metal-3 (0.0)
Metal-4 (0.0)
Metal-5 (0.0)
Metal-6 (0.0)
Polysilicon-1 (0.0)
Polysilicon-2 (0.0)
P-Active (0.0)
New Value: 0
☐ Highlight components
☑ Use Local (cell) LE Settings
☐ Display intermediate capacitances

Logical Effort Gates

A design that is intended to be analyzed with Logical Effort must be composed of special Logical Effort gates. A Logical Effort gate is simply a schematic or layout cell that conforms to the following specifications:

- The cell has an attribute "LEGATE" which is set to "1".
- The cell has only one output, which may have a logical effort attribute (explained below).
- The cell has zero or more inputs/bidirectional ports. Each of these must have a logical effort attribute (explained below).
- The cell has an attribute whose name does not matter, but whose value is "LE.getdrive()", and whose code is set to "Java".



On the input and output exports of the cell, we can define an attribute named "le" (double−click on the export text to get the Export Properties dialog, then click the Attributes button to define the attribute). The value of this attribute is the logical effort of that port. For example, a NAND gate typically has a logical effort on each input of 4/3, and an output logical effort of 2. An inverter is defined to have an input logical effort of 1, and an output logical effort of 1.

The size assigned to the logical effort gate is retrieved via the "LE.getdrive()" call. This value can then be used to size transistors within the gate. The size retrieved is scaled with respect to a minimum sized inverter (as are all other logical effort parameters). So a size of "1" denotes a minimum sized inverter.

While these attributes are defined on the layout or schematic cell *definition*, they must also be present on the instantiated icon or instance of that definition. By default this will be so.

Finally, there must be at least one load that is driven by the gates in order for them to be sized. A load is either a transistor or a capacitor. Gates that do not drive loads, or that do not drive gates that drive loads, will not be assigned sizes.

Logical Effort Libraries

Electric comes with a set of libraries that are specially designed for Logical Effort. Use the **Load Logical Effort Libraries (Purple, Red, and Orange)** command (in menu **Tool / Logical Effort**) to read these libraries.

- The **Purple** library is a set of logic gates that have been tailored for Logical Effort, as described above.
- The **Red** library is a similar set of gates, but they are not setup for Logical Effort. The Red gates can be used in places where Logical Effort is *not* to be done.
- The **Orange** library is a low−level set of gates that is parameterized for a specific fabrication process. Orange gates are used in the Purple and Red libraries, but should not be used elsewhere. The Orange library that comes with Electric is tailored for a generic 180 nanometer process.

Advanced Features

There are several advanced features that may be added to the cell definition:

- Attribute "LEKEEPER=1". This cell is defined as a keeper, whose size will be the size of the smallest Logical Effort gate driving against it, multiplied by the Keeper Ratio.
- Attribute "LEPARALLGRP=0". If set to 0, this gate drives by itself. If an integer greater than zero, all gates with that value whose outputs drive the same network are assumed to drive in parallel. The size needed to drive the load on the network will be equally divided among those gates.
- Attribute "su=−1". This specifies the step−up (fanout) of the gate, and overrides the global fanout specified in the options. If set to −1, this attribute is ignored, and the global value is used.

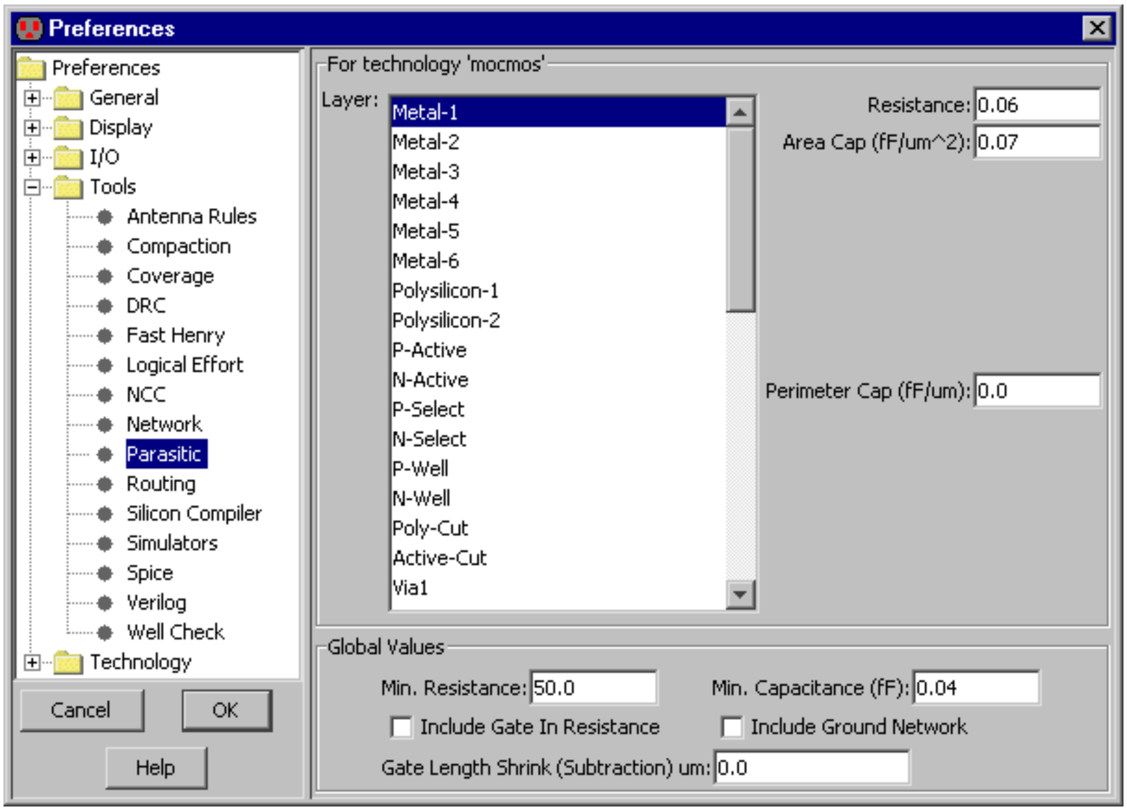
Commands

These commands may be given to the Logical Effort tool (in menu **Tool / Logical Effort**):

- **Optimize for Equal Gate Delays** Optimizes all Logical Effort gates (cells) to have the same delay. The delay is specified by the Global fan−out (step−up) option. This is *NOT* a path optimization algorithm.
- **Optimize for Equal Gate Delays (no caching)** It is intended that both the caching and non−caching algorithms obtain exactly the same result, however due to the difficulty in obtaining and maintaining correctness when it comes to caching, the non−caching algorithm is also available.
- **Print Info for Selected Node** After running sizing, information about a specific logical effort gate can be found by selecting the gate instance and running this command.
- **Back Annotate Wire Loads for the Current Cell** The **Back Annotate Wire Lengths for Current Cell** command runs NCC on the current cell against it's matching layout or schematic cell. Assuming they match, for each LEWIRE in the schematic cell, it finds the half−perimeter of the matching wire in the layout cell (as if the layout was flattened), and then changes the "L" parameter on the LEWIRE to the value. Note, back−annotation is only performed on top level LEWIREs, and it takes into account the wire's length throughout the layout hierarhcy.
- **Clear Sizes on Selected Node(s)** LE sizes are stored as parameters on the LEGATE. Sometimes the sheer number of sizes can overwhelm the allocated process memory, and can also bloat file sizes when they are no longer needed. This command deletes saved sizes on a per−node basis.
- **Clear Sizes in all Libraries** This command deletes saved sizes everywhere.

9−10−1: Parasitic Extraction

The Parasitic Extraction tool is used by netlisters and other parts of the system that need to know about geometric factors. To control Parasitic Extraction, use the "Parasitic" preferences (in menu **File / Preferences...**, "Tools" section, "Parasitic" tab).



For technology 'mocmos'

Layer: Metal-1, Metal-2, Metal-3, Metal-4, Metal-5, Metal-6, Polysilicon-1, Polysilicon-2, P-Active, N-Active, P-Select, N-Select, P-Well, N-Well, Poly-Cut, Active-Cut, Via1

Resistance: 0.06
Area Cap (fF/um^2): 0.07
Perimeter Cap (fF/um): 0.0

Global Values

Min. Resistance: 50.0
Min. Capacitance (fF): 0.04
☐ Include Gate In Resistance
☐ Include Ground Network
Gate Length Shrink (Subtraction) um: 0.0

Each layer of the current technology is listed, and you can set its unit resistance, area capacitance, and edge capacitance. In addition, you can set minimum resistance and capacitance values for the entire technology.

The bottom section controls values for the entire technology. You can set the minimum resistance and capacitance for the entire technology. The "Gate Length Shrink" is a compensation factor for gate lengths. Some process technologies shrink the gate length by a fixed amount. "Include Gate In Resistance" requests that a transistor's gate area be included in overall area calculations for resistance determination. "Include Ground Network" requests that ground networks be analyzed.

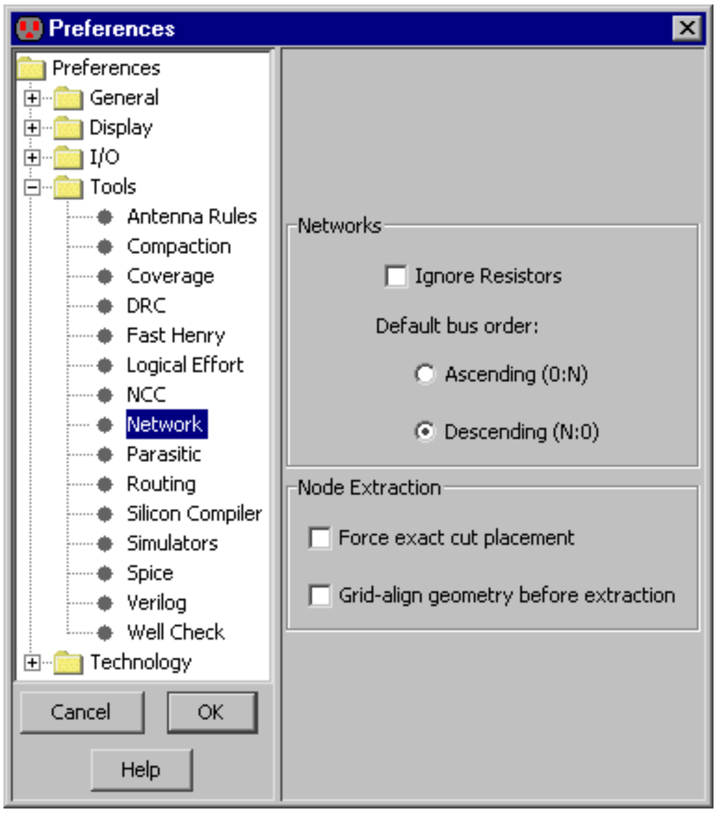
9−10−2: Node Extraction

Because Electric captures connectivity information during design, there is no need for "node extraction", the process of extracting connectivity from layout. However, there are situations where a circuit has only layout and no connectivity, specifically when a circuit has been read into Electric from CIF, GDS, or other formats that have no connectivity information in them.

When CIF, GDS, and other foreign file formats are read into Electric, the cells they create are composed entirely of pure−layer nodes (see Section 6−10−1). These nodes appear to represent the circuit correctly, and can even be written back out to CIF or GDS correctly. But the missing connectivity information means that Electric cannot properly analyze these circuits (cannot do DRC, simulation, etc.)

The solution is to convert this geometry into properly connected components. To convert the current cell into connected geometry, use the **Extract Current Cell** command (from menu **Tool / Network**). To convert the current cell and all subcells, use the **Extract Current Hierarchy** command. Electric creates new versions of the layout cells that have higher−level nodes and arcs in them.

Although the process of converting layout into connectivity information is difficult, it can usually be done correctly. In Electric, this process is complicated by the fact that the resulting connectivity information must be expressed as a set of "high−level" primitives (transistors and contacts) which have their own ways of appearing in the layout. Therefore, it is not always possible to extract layout precisely. For example, if the design rules for a transistor require that polysilicon extend beyond the gate area by 2 units, the transistor primitive for that technology will have this extra geometry built into it. But what would happen if the geometry to be extracted extends by 3 units? Electric adds an extra 1−unit arc to fill−out the extra geometry that it finds. Worse yet, what would happen if the geometry extends by only 1 unit? Electric simply cannot represent this with its primitives. It will create the transistor, but it will no longer match the original geometry. In general, the system attempts to create high−level primitives that mimic the original geometry. It often leaves small pure−layer nodes behind to complete the extraction. As an aid in debugging the extraction process, these extra pure−layer nodes are highlighted in the resulting cell.



Networks

☐ Ignore Resistors

Default bus order:

○ Ascending (0:N)

[✓ Descending (N:0)]

Node Extraction

☐ Force exact cut placement

☐ Grid-align geometry before extraction

Two "Network" preferences are available to control the extraction process (in menu **File / Preferences...**, "Tools" section, "Network" tab).

The first is "Force exact cut placement", which requires that the cut (or via) locations appear exactly in the same place once extracted. Without this preference, Electric will find contact areas and replace them with contact nodes regardless of where the contact nodes place the cuts. With this preference selected, Electric will place contact nodes in such a way that the cut layers land in the correct original locations.

The disadvantage of forcing exact cut placement is that Electric will create many contact nodes, one for each cut layer. In multi−cut situations, this may be many more nodes than are necessary.

The other preference is "Grid−align geometry before extraction". Doing this makes the extraction process run more smoothly, but may cost slightly in accuracy.

Chapter 9: Tools

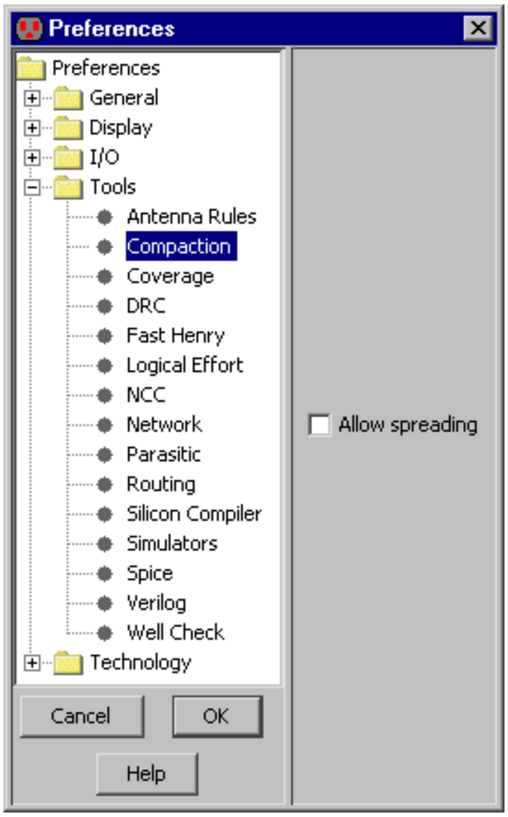
9−11: Compaction

The compaction tool squeezes layout down to minimal design−rule spacing. It does this by doing single−axis compaction, alternating horizontal and vertical directions until no further space can be found. Each pass of compaction squeezes either to the left or to the bottom of the circuit.

To compact, use the **Do Compaction** command (in menu **Tool / Compaction**).

The "Compaction" preferences (in menu **File / Preferences...**, "Tools" section, "Compaction" tab). can tell the compactor to expand the circuit if it is too close for the design rules.

For an example of compaction, open the Samples library and edit the cell "tool−Compaction" (you can read the library with the **Load Library** command, in menu **Help / Samples**).



☐ Allow spreading

Be warned that the compaction tool is experimental and doesn't always achieve optimal results.

9−12: Silicon Compiler

Electric has a silicon compiler called QUISC (the Queen's University Interactive Silicon Compiler). It is a powerful tool that can do placement and routing of standard cells from a schematic or a structural VHDL description. When placing and routing VHDL, it is compiled into a netlist which is then used to drive placement and routing. When placing and routing a schematic, it is converted into VHDL, compiled to a netlist, and laid−out. Thus, a byproduct of silicon compilation will be a {net.quisc} view of a cell, and potentially a {vhdl} view.

Be warned that the silicon compiler is rather old, and so it produces layout that alternates standard cell rows and routing rows. Modern silicon compilers use multiple metal processes to route over the standard cells, but this system does not. This system uses two layers: a *vertical* routing arc to run in and out of cells, and a *horizontal* routing arc to run between the cells in the routing channel. It also uses *power* arcs to bring power and ground to the cell rows, and *main power* arcs to connect the rails on the left and right.

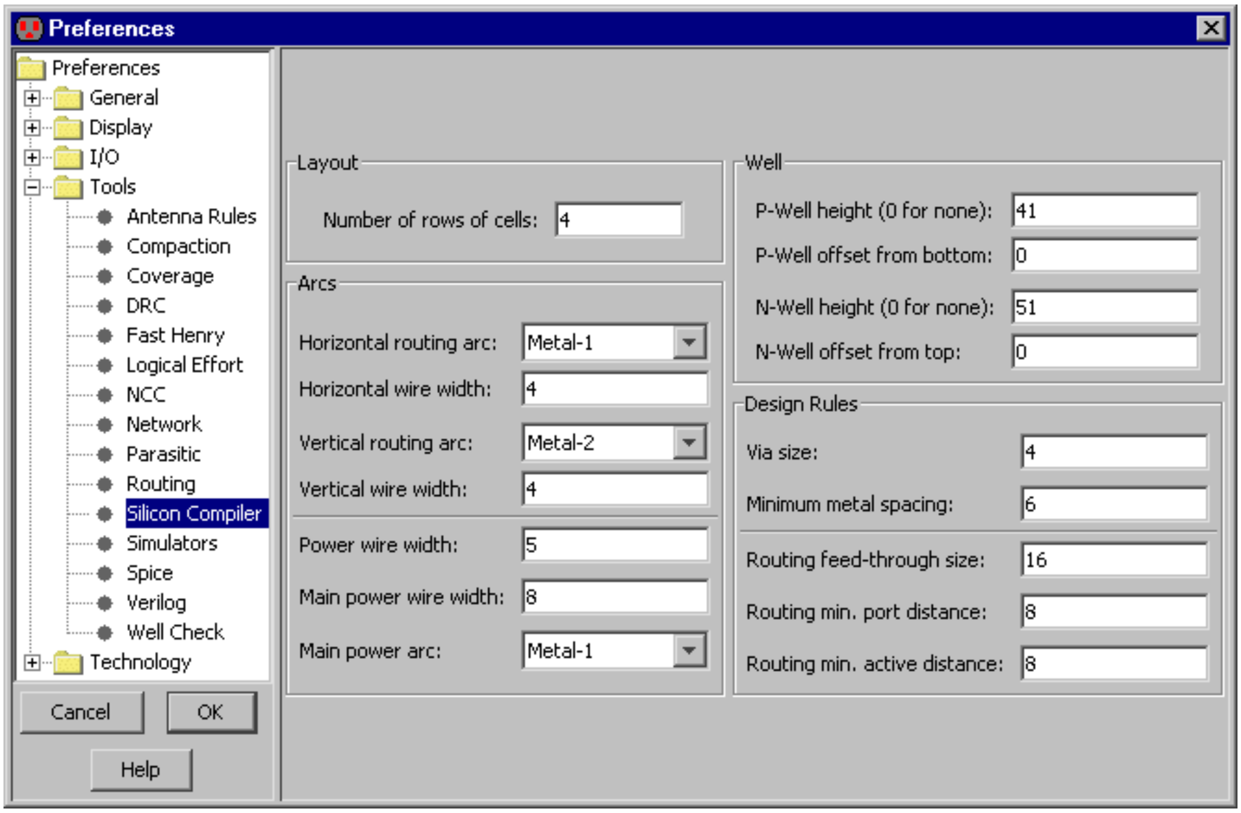
The VHDL description is normally placed in the "vhdl" view of a cell (see Section 4−10 for more on text editing). There is a VHDL example in cell "tool−SiliconCompiler{vhdl}" of the "samples" library. To access it, use the **Load Library** command (in menu **Help / Samples**).

To convert a schematic or VHDL cell into layout, use the **Convert Current Cell to Layout** command (in menu **Tools / Silicon Compiler**). To compile VHDL to the {net.quisc} view, use the **Compile VHDL to Netlist View** command (this is typically not needed, since the previous command does it automatically).

When creating a schematic or VHDL cell to be compiled, it is important to know what primitives are available in the standard cell library. Electric comes with a CMOS cell library in the MOSIS CMOS ("mocmos") technology. This library is not correct, and exists only to illustrate the Silicon Compiler. These component declarations are available:

```
component and2 port(a1, a2 : in bit; y : out bit);  end component;
component and3 port(a1, a2, a3 : in bit; y : out bit);  end component;
component and4 port(a1, a2, a3, a4 : in bit; y : out bit);  end component;
component inverter port(a : in bit; y : out bit);  end component;
component nand2 port(a1, a2 : in bit;  y : out bit);  end component;
component nand3 port(a1, a2, a3 : in bit;  y : out bit);  end component;
component nand4 port(a1, a2, a3, a4 : in bit; y : out bit);  end component;
component nor2 port(a1, a2 : in bit; y : out bit);  end component;
component nor3 port(a1, a2, a3 : in bit;  y : out bit);  end component;
component nor4 port(a1, a2, a3, a4 : in bit;  y : out bit);  end component;
component or2 port(a1, a2 : in bit; y : out bit);  end component;
component or3 port(a1, a2, a3 : in bit;  y : out bit);  end component;
component or4 port(a1, a2, a3, a4 : in bit;  y : out bit);  end component;
component rdff port(d, ck, cb, reset : in bit; q, qb : out bit);  end component;
component xor2 port(a1, a2 : in bit;  y : out bit);  end component;
```

The "Silicon Compiler" preferences (in menu **File / Preferences...**, "Tools" section, "Silicon Compiler" tab) let you control many aspects of placement and routing.



Layout
Number of rows of cells: 4

Arcs
Horizontal routing arc: Metal-1
Horizontal wire width: 4
Vertical routing arc: Metal-2
Vertical wire width: 4
Power wire width: 5
Main power wire width: 8
Main power arc: Metal-1

Well
P-Well height (0 for none): 41
P-Well offset from bottom: 0
N-Well height (0 for none): 51
N-Well offset from top: 0

Design Rules
Via size: 4
Minimum metal spacing: 6
Routing feed-through size: 16
Routing min. port distance: 8
Routing min. active distance: 8

- The "Layout" section controls the number of rows of cells that will be created. A one−row circuit may be exceedingly wide and short, so you may wish to experiment with this value. For a square circuit, the number of rows should be the square root of the number of instances in the circuit (the number of instances appears as the sum of the unresolved references, listed by the VHDL Compiler).
- The "Arcs" section lets you set the horizontal and vertical routing arcs, as well as the power rails.
- The "Well" section gives you the option of placing blocks of P−well and N−well over the cell rows.
- The "Design Rules" section lets you control Via size, metal spacing, feed−through size, port distance, and active distance.

Chapter 10: The JELIB File Format

10−1: JELIB File Format

This chapter describes Electric's native file format, which ends in "jelib". The earlier file format, which ends in "elib", remains undocumented and is no longer recommended.

JELIB files are text−readable files. Each line of a JELIB file starts with an identifying character that distinguishes the line. Blank lines, and those that start with the comment identifying character (#) are ignored. There is no limit to the length of a line of text.

After the identifying character at the start of a line, there are a set of fields. All of the fields are separated by the separator character (|) except for the first field, which begins immediately after the identifying character. No blank spaces are allowed on a line (that is, any blank spaces are treated as valid characters). Control characters (such as the identifying characters) must be upper case. In order to insert a '|' or '\n' or '\r' into a field, it must be enclosed in the quotation mark characters ("). Backslash character can be used inside enclosed strings to denote special characters:

Characters	Meaning
\n	line feed character (\n)
\r	carriage return character (\r)
\"	quotation mark character (")
\\	backslash character (\)

Each of the different types of lines in the file has a fixed set of fields that must appear. Some line types also allow additional fields at the end to add variables (attribute/value pairs, see Section 10−4−1).

The JELIB file has 3 parts: the header, cells, and trailer.

The header has these elements:

H	Header information; variable fields are allowed
V	View information
L	External library information
R	External cell in the above external library

F	External export in the above external cell
T	Technology information; variable fields are allowed
D	Primitive Node information in the above technology
P	Primitive Port information in the above primitive node
W	Primitive Arc information in the above technology
O	Tool information

The cells have these elements:

C	Cell header; variable fields are allowed
N	Primitive node information in the current cell; variable fields are allowed
I	Cell instance information in the current cell; variable fields are allowed
A	Arc information in the current cell; variable fields are allowed
E	Export information in the current cell; variable fields are allowed
X	Cell termination

The trailer has this element:

G	Group information

Everything in the file is completely ordered. There is an ordering to the external libraries, cells in those libraries, technologies, tools, cells, nodes/arcs/exports in a cell, etc. Even the extra variables on a line are ordered. The ordering is usually a name sort. By ordering everything in the file, the exact same file is generated every time, and source−code comparison operations will accurately find differences between two files. Note, however, that the JELIB reader does not require any sorting, and can handle the data in any order.

10−2−1: Header, View, and Tool

Headers

The first line in the JELIB file should be the "H" header line. The syntax is:

H<name> | <version> [| <variable>]*	
<name>	the name of the library.
<version>	the version of Electric that wrote the library.
<variable>	a list of variables on the library (see Section 10−4−1).

The name of the library is used in the JELIB file to identify references to this library. The actual name of this library is obtained from the file path of this JELIB file.

Example:

```
    Hlatches|8.01
```

Declares that library "latches" was written from Electric version 8.01.

Views

All views used in the library must be declared.

V<full name> | <name>	
<full name>	the full name of the view.
<name>	the abbreviation name of the view.

Example:

```
    Vlayout|lay
```

Declares view with abbreviation name "lay" and full name "layout".

Tools

There is no need to declare all tools in the header. The only reason for a tool declaration to exist is if the tool has preferences variables stored on it. If there are multiple tool lines, they are sorted by the tool name. The syntax is:

O<name> [| <variable>]*	
<name>	the name of the tool.
<variable>	a list of preferences on the tool (stored as variables, see Section 10−4−1).

Example:

```
    Osimulation|SpiceEngine()I2|SpiceLevel()I1
```

Declares two preferences on the “simulation” tool object. The first is called “SpiceEngine” and is set to the integer value 2. The second is called “SpiceLevel” and is set to the integer value 1.

10−2−2: External References

After the header line, all external libraries cells and exports must be declared. This allows the file reader to quickly find all libraries that will be needed for the design, and to reconstruct any missing cells and exports. The cells are listed under their libraries. The exports are listed under their cells. If there are multiple external library lines, they are sorted by library name; where there are multiple external cells in a library, they are sorted by their name; and where there are multiple external exports in a cell, they are sorted by their name.

The syntax of an external library reference is:

L<name> | <path>	
<name>	the name of the external library.
<path>	the full path to the disk file with the library.

The name of the library is used in JELIB file to references to this library. The actual name of this library is obtained from the path.

The syntax of an external cell reference is:

R<name> | <lowX> | <highX> | <lowY> | <highY> | <creation> | <revision>	
<name>	the name of the external cell.
<lowX>	the low X bounds of the cell contents.
<highX>	the high X bounds of the cell contents.

<lowY>	the low Y bounds of the cell contents.
<highY>	the high Y bounds of the cell contents.
<creation>	the creation date of the cell (Java format).
<revision>	the revision date of the cell (Java format).

The Java format for dates (the creation and revision dates) is in milliseconds since the "epoch" (Midnight on January 1, 1970, GMT).

The syntax of an external export reference is:

F<name> | <centerX> | <centerY>	
<name>	the name of the external export.
<centerX>	the X coordinate of the center of export polygon.
<centerY>	the Y coordinate of the center of export polygon.

Examples:

```
    Lspiceparts|/home/strubin/electric/spiceparts.jelib
    Rgate;1{sch}|-4|4|0|2|1092185029000|1092185060000
    Fout|0|2
```

Declares that an external library called "spiceparts" will be used by the current library, and that it can be found at "/home/strubin/electric/spiceparts.jelib". In that library is a cell called "gate;1{sch}" whose contents run from –4 to 4 in X and 0 to 2 in Y. In that cell is an export called "out" with center at (0,2).

10−2−3: Technologies

Technologies

All technologies used in the library must be in the header. The other reason for a technology declaration to exist is if the technology has preferences stored on it. If there are multiple technology lines, they are sorted by technology name. The syntax is:

T<name> [| <variable>]*	
<name>	the name of the technology.
<variable>	a list of preferences on the technology (stored as variables, see Section 10−4−1).

Examples:

```
    Tmocmos
```

Declares that there should be a technology called "mocmos".

```
    Tmocmos|ScaleFORmocmos()D200
```

Declares the technology "mocmos" and also creates a preference on that technology object called "ScaleFORmocmos" which is a double−precision value equal to 200.

10−3−1: Cells

After the header information, each cell is described. A cell consists of a cell declaration ("C") followed by a number of node ("N"), arc ("A"), and export ("E") lines. The cell is terminated with a cell−end line ("X"). Inside of a cell, all nodes come first and are sorted by the node name; arcs come next and are sorted by the arc name; finally come exports, sorted by the export name. Also, when there are multiple cells, their appearance in the file is sorted by the cell name. The syntax is:

C<name> | <tech> | <creation> | <revision> | <flags> [| <variable>]*	
<name>	the name of the cell in the form "NAME;VERSION{VIEW}".
<tech>	the technology of the cell.
<creation>	the creation date of the cell (Java format).
<revision>	the revision date of the cell (Java format).
<flags>	flags for the cell.
<variable>	a list of variables on the cell (see Section 10−4−1).

The Java format for dates (the creation and revision dates) is in milliseconds since the "epoch" (Midnight on January 1, 1970, GMT).

The <flags> field consists of any of the following letters, (sorted alphabetically):

"C" if this cell is part of a cell−library.
"E" if the cell should be created "expanded".
"I" if instances in the cell are locked.
"L" if everything in the cell is locked.
"T" if this cell is part of a technology−library.

Example:

```
    CrxArray;1{lay}|mocmos|1092185029000|1092185060000|I
```

Declares cell "rxArray{lay}", version 1, associated with the "mocmos" technology. The cell was created at date 1092185029000 and last modified at date 1092185060000. All instances in the cell are locked.

10−3−2: Node Instances

Inside of a cell definition, node instances are declared with the "N" and "I" lines. "N" is for primitive nodes and "I" is for cell instances. All nodes are sorted by the node name. The syntax is:

N<type> | <name> | <nameTD> | <x> | <y> | <width> | <height> | <orientation> | <flags> [| <variable>>]*	
I<type> | <name> | <nameTD> | <x> | <y> | <orientation> | <flags> | <TD>> [| <variable>]*	
<type>	the type of the node instance. For primitive node instances, this has the form: [<technology>:]<primitive−node>. If <technology> is omitted, the technology of the cell is assumed. For cell instances, it has the form: [<library>:]<cell>;<version>{<view>}. If <library> is omitted, the library defined by this JELIB file is assumed.
<name>	the name of the node instance. If there are multiple nodes with the same name, then the name is quoted, and a unique integer follows the close quote.
<nameTD>	a text descriptor for the name (when displayed).
<x>	the X coordinate of the anchor point of the node instance.
<y>	the Y coordinate of the anchor point of the node instance.
<width>	the width of the primitive node (must be non−negative).
<height>	the height of the primitive node (must be non−negative).
<orientation>	the orientation of the node (see below).
<flags>	flags for the node instance (see below).
<TD>	a text descriptor for the cell instance name (does not apply to primitives).
<variable>	a list of variables on the node instance (see Section 10−4−1).

The <orientation> field consists of any of the following letters, with an optional numeric part at the end:

"X" if the node instance is X−mirrored (mirrored about Y axis).
"Y" if the node instance is Y−mirrored (mirrored about X axis).
"R" each letter rotates the node instance at 90 degrees counter−clockwise.
Num Any digits at the end are additional rotation in tenths of a degree.

The <flags> field consists of any of the following letters, sorted alphabetically, with the numeric part at the end:

"A" if the node instance is hard−to−select.
"E" if the node instance is "expanded".
"L" if the node instance is locked.
"V" if the node instance is visible only inside the cell.
"W" if the node instance is wiped (covered by an arc, so no need to draw).
Num Any digits at the end are the technology−specific bits.

Examples:

```
    Nschematic:Transistor|mos@0||2|0|4|4|R|2|ATTR_length(D5G0.5;X-0.5;Y-1;)S2
```

Places a schematic Transistor called "mos@0" at (2,0), size 4x4, rotated 90 degrees. The flag field "2" is numeric, and therefore is technology−specific information (in this case, it makes the transistor be pMOS).

There is one attribute on the node, called "length", with the value "2" (a string). This attribute is displayed, anchored at its center ("D5"), is 1 half grid unit in size ("G0.5;"), and is offset (−0.5, −1) from the node center ("“X−0.5;Y−1;").

```
Ilow;1{lay}|"HAPPY"1||14|12|Y|E|D5G4;
```

Places an instance of cell "low{lay}" from the library defined in this JELIB file. The instance is named "HAPPY" (the name field is "HAPPY"1 which means this is the first instance with this name). It is at (14,12),mirrored in Y, and is rotated 0. The "E" means that the node is expanded, but when it is unexpanded, its name is described by D5G4; (D5 means a centered anchor point, G4; means size 4 units).

10−3−3: Arc Instances

Inside of a cell definition, arc instances are declared with the "A" line. All arcs are sorted by the arc name. The syntax is:

A<type> | <name> | <nameTD> | <width> | <flags> | <headNode> | <headPort> | <headX> | <headY>> | <tailNode> | <tailPort> | <tailX> | <tailY>> [| <variable>]*	
<type>	the type of the arc instance. It has the form: [<technology>:]<arc>. If technology is omitted, the technology of the cell is assumed.
<name>	the name of the arc instance.
<nameTD>	a text descriptor for the name (when displayed).
<width>	the width of the arc instance.
<flags>	flags for the arc instance (see below).
<headNode>	the name of the node at the head of the arc instance.
<headPort>	the name of the port on the head node (may be blank if there are no choices).
<headX>	the X coordinate of the head of the arc instance.
<headY>	the Y coordinate of the head of the arc instance.
<tailNode>	the name of the node at the tail of the arc instance.
<tailPort>	the name of the port on the tail node (may be blank if there are no choices).
<tailX>	the X coordinate of the tail of the arc instance.
<tailY>	the Y coordinate of the tail of the arc instance.
<variable>	a list of variables on the arc instance (see Section 10−4−1).

The <flags> field consists of any of the following letters, sorted alphabetically, with the numeric part at the end:

"A" if the arc instance is hard−to−select.
"B" if the arc instance has an arrow line on the body (use "X" and "Y" for arrow heads).
"F" if the arc instance is NOT fixed−angle (fixed−angle is more common).
"G" if the arc instance has its head connection negated.
"I" if the arc instance has its head NOT extended.

"J" if the arc instance has its tail NOT extended.
"N" if the arc instance has its tail connection negated.
"R" if the arc instance is rigid.
"S" if the arc instance is slidable.
"X" if the arc instance has an arrow on the head (use "B" for an arrow body).
"Y" if the arc instance has an arrow on the tail (use "B" for an arrow body).
Num Any digits at the end are the angle of the arc (in tenths of a degree).

Examples:

```
AMetal−1|net@0||3|S1800|contact@0||10|10|pin@0||20|10
```

Places a metal−1 arc (from the technology of the cell). The arc is named "net@0", is 3 wide, slidable, and at a 180 degree angle. The arc runs from (10,10) on node "contact@0", to (20,10) on node "pin@0".

```
Aschematic:bus|net@161||1|IJ2700|busHat@4|s[1:8]|42|14|conn@15|y|42|25
```

Places a bus arc (from schematic) named "net@161"", width 1, not end−extended on either end, at 270 degrees angle. The bus runs from (42,14) on node busHat@4 (port "s[1:8]") to (42,25) on node "conn@15" (port "y").

10−3−4: Exports

Inside of a cell definition, exports are declared with the "E" line. All exports are sorted by their name. The syntax is:

E<name> | <TD> | <originalNode> | <originalPort> | <flags> | [| <variable>]*	
<name>	the name of the export.
<TD>	the text descriptor for writing the port (described later).
<originalNode>	the name of the node instance in this cell that the export resides on.
<originalPort>	the name of the port on the exported node instance (may be blank if there are no choices).
<flags>	flags for the export (see below).
<variable>	a list of variables on the export (see Section 10−4−1).

The <flags> field has the format:

<characteristics> [/A] [/B]

<characteristics> the nature of the export. Choose from the following:

"U" unknown.
"I" input.
"O" output.
"B" bi−directional.

"P" power.
"G" ground.
"C" clock.
"C1" clock phase 1.
"C2" clock phase 2.
"C3" clock phase 3.
"C4" clock phase 4.
"C5" clock phase 5.
"C6" clock phase 6.
"RO" reference output.
"RI" reference input.
"RB" reference base.
/A indicates that the export is always drawn
/B indicates that the export is body−only (no equivalent on the icon)

Example:

```
    Es[18]||conn@14|a|D5G2;|I/B
```

Exports port "a" of node instance "conn@14"” and calls it "s[18]". The text of the export is attached at the center of the port ("D5") and is 2 units high ("G2;"). It is of type input, and only appears in the contents (not the icon).

10−4−1: Variables

Variables may be attached to any object in the Electric database. They appear at the end of many of the lines in the file. When more than 1 variable is listed on an object, they are sorted by the variable name. The syntax is:

<name> (<TD>) <type> <value>	
<name>	the name of the variable.
<TD>	the text descriptor (when the variable is visible).
<type>	the type of data attached.
<value>	the data. If it starts with "[", it is an array of the form [, , …]

<name> and <value> fields may be enclosed in quotation marks. Backslash character can be used inside enclosed strings to denote special characters.

The <type> field can be one of these: "B" Boolean ("T" or "F")
"C" Cell (of the form :).
"D" Double.
"E" Export (of the form : :).
"F" Float.
"G" Long.
"H" Short.
"I" Integer.
"L" Library name.
"O" Tool name.
"P" Primitive Node prototype (of the form :).
"R" Arc prototype (of the form :).
"S" String.
"T" Technology name.
"V" Point2D (of the form <x> / <y>).
"Y" Byte (0−255).

Examples:

```
    ART_message(D5G8;)StxArray4x4B
```

Adds a variable called "ART_message" with the string "txArray4x4B". The text descriptor indicates centered text ("D5") that is 8 units tall ("G8;").

```
    ART_degrees()F[0.0,3.1415927]
```

Adds a variable called "ART_degrees" with an array of 2 floating point values: 0.0 and 3.1415927.

```
EXPORTS()E[ccc:gate;1{sch}:a,"ccc:hate;1{sch}:b[0:4]"]
```

Adds a variable called "EXPORTS" with an array of 2 exports of the cell "ccc:gate;1{sch}": "a" and "b[0:4]".

```
ATTR_z0(D5G0.5;NPY1;)I50
```

Adds an attribute called "z0" with the integer value 50. It is displayed anchored at the center ("D5"), 0.5 unit tall ("G0.5;"), written as "name=value" ("N"), is a parameter ("P"), and is offset by 1 in Y ("Y1;").

10−4−2: Text Descriptors

Text descriptors appear in every Variable, and also in other places (cell instances and exports). It consists of these fields:

A <size> ;	Text is absolute size (in points).
B	Text is bold.
C <color> ;	Text is drawn in the color index given.
D0	Text is anchored at its center, limited to the size of its owner.
D1	Text is anchored at its lower−left.
D2	Text is anchored at its bottom.
D3	Text is anchored at its lower−right.
D4	Text is anchored at its left.
D5	Text is anchored at its center.
D6	Text is anchored at its right.
D7	Text is anchored at its upper−left.
D8	Text is anchored at its top.
D9	Text is anchored at its upper−right.
F <font> ;	Text is shown in the named font.
G <size> ;	Text has relative size (in grid units).
H	Variable is inheritable (only for variables on Cells or Exports).
I	Text is italic.
L	Text is underlined.
N	Variable is written in the form "NAME=VALUE".
OJ	Text is Java code.
OL	Text is Lisp code.
OT	Text is TCL code.
P	Variable is a parameter.
R	Text is rotated 90 degrees.
RR	Text is rotated 180 degrees.
RRR	Text is rotated 270 degrees.

T	Text is interior (seen only when inside the cell).
UR	Value is in Resistance units.
UC	Value is in Capacitance units.
UI	Value is in Inductance units.
UA	Value is in Current units.
UV	Value is in Voltage units.
UD	Value is in Distance units.
UT	Value is in Time units.
X <xoff> ;	Text is offset in X from object center.
Y <yoff> ;	Text is offset in Y from object center.

Example:

```
D4G8;
```

The text is anchored on the left ("D4") and is 8 units tall ("G8;").

10−4−3: Groups

After all of the cells are listed, they are organized into groups. Each group line consists simply of a list of cells in that group. The first cell listed is the “main schematics” of the group. If there is no such cell, the first field is empty. After that, the cells appear in alphabetical order.

When multiple groups are declared, they appear sorted by the group name (which is derived from the cell names in it). The syntax is:

G<cell> | <cell> | … | <cell>	
<cell>	the name of the cells in the group. <cell> may consists only of proto name, because all cells with the same base name are put into the same group.

Examples:

```
Gsam;2{sch}
```

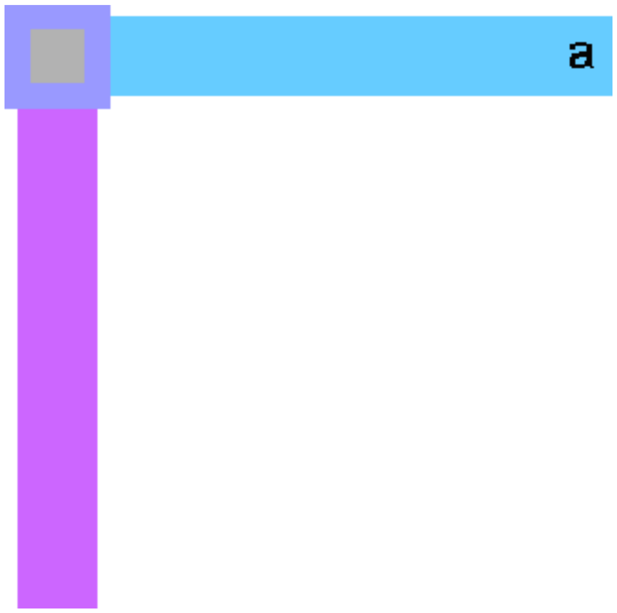
"sam;2{sch}" is the main schematic of the cell group consisting of all views and versions of cells with proto name "sam".

```
G|high|higha
```

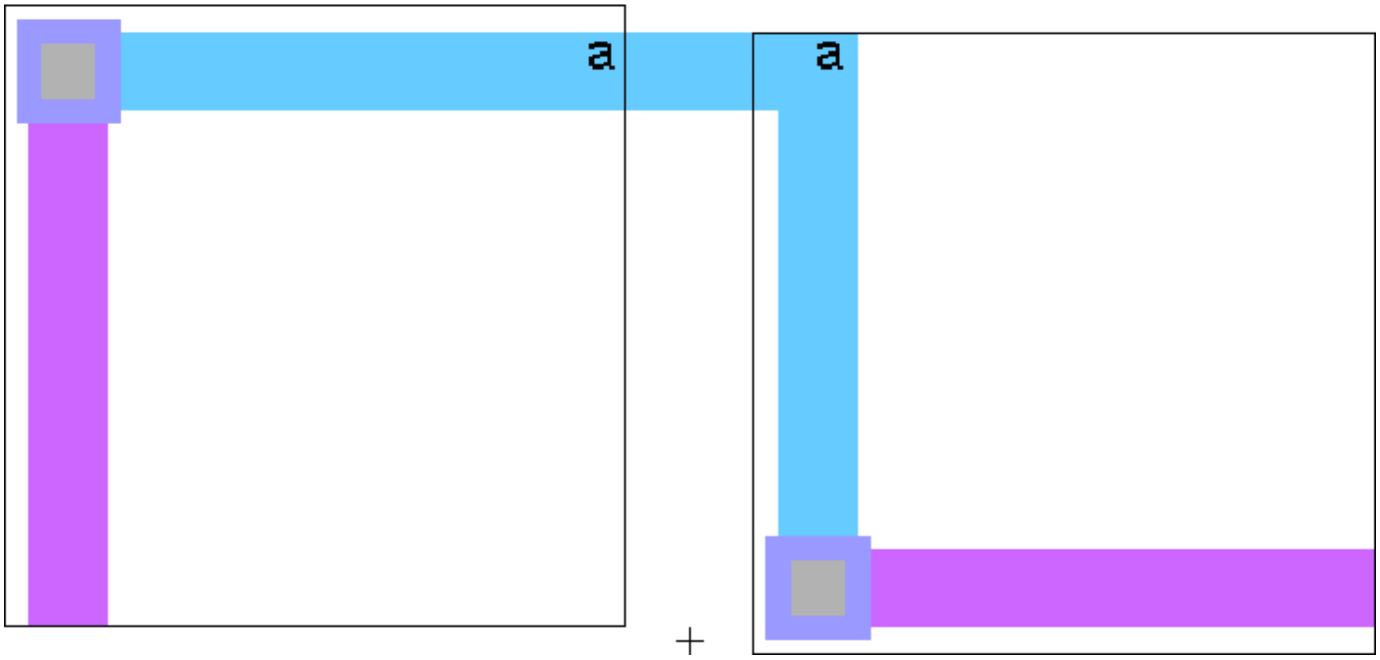
Places all views and versions of cells with proto names "high" and "higha" in the same group (there is no main schematic).

10−4−4: Example

As an example of the JELIB format, let us assume a design with two levels of hierarchy. The bottom level of hierarchy (cell "low") has 3 nodes, two arcs, and an export:



The top level of hierarchy (cell "high") has two instances of the cell (the right instance is rotated 90 degrees) and an arc connecting them:



Here is the JELIB file for the above layout.

```
# header information:
Hccc|8.02

# Views:
Vlayout|lay

# Technology mocmos
Tmocmos|MoCMOSNumberOfMetalLayers()I6
```

```
# Cell high{lay}
Chigh;1{lay}|mocmos|1093555876000|1094258888640|
Ngeneric:Facet-Center|art@0||0|0|0|0||AV
Ilow;1{lay}|low@0||-14|12||E|D5G4;
Ilow;1{lay}|low@1||15|12|R|E|D5G4;
AMetal-1|net@0||3|S0|low@1||5|22|low@0||-4|22
X

# Cell low{lay}
Clow;1{lay}|mocmos|1093555232000|1094258870406|
Ngeneric:Facet-Center|art@0||0|0|0|0||AV
NMetal-1-Metal-2-Con|contact@0||-10|10|5|5||
NMetal-1-Pin|pin@0||10|10|3|3||W
NMetal-2-Pin|pin@1||-10|-10|3|3||W
AMetal-1|net@0||3|S1800|contact@0||-10|10|pin@0||10|10
AMetal-2|net@1||3|S900|contact@0||-10|10|pin@1||-10|-10
Ea|D5G2;|pin@0||I
X

# Groups:
G|high;1{lay}
G|low;1{lay}
```