

Using the Electric VLSI Design System

Steven M. Rubin

Table of Contents

Using the Electric™ VLSI Design System.....	1
Chapter 1: INTRODUCTION.....	3
1-1: Welcome.....	3
1-2: Requirements.....	4
1-3: UNIX Installation.....	5
1-4: Macintosh Installation.....	9
1-5: Windows Installation.....	11
1-6: Fundamental Concepts.....	14
1-7: The Display.....	17
1-8: The Mouse.....	19
1-9: The Keyboard.....	21
Quick Keys.....	21
The Interrupt Key.....	22
Don't Type This Key.....	22
1-10: IC Layout Example.....	23
1-11: Schematics Example.....	28
Chapter 2: BASIC EDITING.....	33
2-1: Selection.....	33
Selecting Nodes and Arcs.....	33
Selection Appearance.....	33
Selecting Areas.....	34
Selecting Text.....	34
Controlling Selection.....	34
Easy and Hard Selection.....	35
2-2: Circuit Creation.....	36
Node Creation.....	36
Arc Creation.....	36
Special Cases.....	38
2-3: Circuit Deletion.....	39
2-4: Circuit Modification.....	41
Movement.....	41
Other Modification.....	42
2-5: Changing Size.....	45
Node Sizing.....	45
Arc Sizing.....	46
2-6: Changing Orientation.....	47
Chapter 3: HIERARCHY.....	49
3-1: Cells.....	49
3-2: Creating and Deleting Cells.....	50
Cell Creation and Deletion.....	50
3-3: Creating Instances.....	52
3-4: Examining Instances.....	54
3-5: Moving Up and Down the Hierarchy.....	55
3-6: Exports.....	57
Export Creation.....	57
Export Information.....	59



Table of Contents

Export Deletion and Movement.....	60
3-7: Cell Information.....	61
Miscellaneous Commands.....	61
Cell Options.....	63
The Cell Explorer.....	64
3-8: Rearranging Hierarchy.....	66
Creating New Levels of Hierarchy.....	66
Removing Levels of Hierarchy.....	66
3-9: Libraries.....	67
Reading Libraries.....	67
Writing Libraries.....	68
Standard-Cell Libraries.....	70
3-10: Copying Between Libraries.....	71
3-11: Cell Views.....	73
Setting a Cell's View.....	73
Switching between Views of a Cell.....	74
Creating and Deleting Views.....	74
3-12: Automatic View Generation.....	75
Conversion between Layout and Schematic.....	75
Skeletonization.....	75
Icons.....	75
VHDL.....	76
Chapter 4: THE DISPLAY.....	77
4-1: Introduction to the Display.....	77
4-2: The Messages Window.....	78
4-3: Creating and Deleting Editing Windows.....	79
Multiple Editing Windows.....	79
Splitting Editing Windows.....	79
4-4: Scaling and Panning.....	81
Scaling.....	81
Panning.....	81
Saving Views.....	82
4-5: Layer Visibility.....	83
4-6: Colors.....	84
Electric's Color Model.....	84
Editing Colors.....	84
Setting the Color and Pattern of Layers.....	86
4-7: Grids and Alignment.....	87
Drawing a Grid.....	87
Aligning to a Grid.....	88
Aligning to Objects.....	88
Measuring.....	88
4-8: The Component Menu.....	90
4-9: Hardcopy.....	91
4-10: Text Windows.....	93
4-11: 3D Display.....	95
Chapter 5: WIRE PROPERTIES.....	97



Table of Contents

5-1: Introduction to Arcs.....	97
5-2: Constraints.....	98
Rigid and Fixed-Angle Arcs.....	98
Slidable Arcs.....	99
Constraint Propagation.....	100
5-3: Setting Constraints.....	101
5-4: Other Arc Properties.....	102
Directionality.....	102
Negation.....	102
End Extension.....	103
Naming.....	103
Curvature.....	103
5-5: Default Arc Properties.....	104
Chapter 6: ADVANCED EDITING.....	105
6-1: Making Copies.....	105
Duplication.....	105
Cut-and-Paste.....	105
6-2: Creation Defaults.....	106
6-3: Options.....	108
6-4: Making Arrays.....	110
6-5: Spreading Circuitry.....	112
6-6: Replacing Circuitry.....	113
Special Considerations.....	114
6-7: Undo Control.....	115
6-8: Text.....	116
Understanding Text.....	116
Selecting Text.....	117
Modifying Text.....	118
Text Defaults.....	120
Text Attributes.....	121
Cell Parameters.....	123
6-9: Networks.....	125
Naming Networks.....	126
Bus Naming.....	127
Power and Ground.....	127
Global Networks.....	128
6-10: Outline Editing.....	129
What is an Outline?.....	129
Manipulating Outlines.....	130
Special Outline Generation.....	130
6-11: Project Management.....	132
Creating a new Project.....	132
Checking Cells In and Out.....	133
Under the Hood.....	134
6-12: Emergencies.....	135
Database Corruption.....	135
Running out of Memory.....	135
Crash Recovery.....	135



Table of Contents

Chapter 7: DESIGN ENVIRONMENTS.....	137
7-1: Technologies.....	137
Many Different Technologies.....	137
What is in a Technology.....	137
Controlling Technologies.....	138
7-2: Units.....	139
Lambda.....	139
Display Units.....	141
Internal Units.....	141
7-3: I/O Specifications.....	142
CIF Control.....	142
GDS Control.....	144
EDIF Control.....	145
DEF Control.....	146
CDL Control.....	146
DXF Control.....	146
SUE Control.....	147
7-4: The MOS Technologies.....	148
7-5: The MOSIS CMOS Technology.....	150
7-6: The Schematic Technology.....	152
Digital Schematics.....	152
Analog Schematics.....	153
Multipage Schematics and Frames.....	154
7-7: The Artwork Technology.....	155
7-8: The FPGA Technology.....	157
Primitive Definition Section.....	157
Block Definition and Architecture Sections.....	159
Commands.....	162
7-9: The Generic Technology.....	163
Special Arcs.....	163
Special Nodes.....	163
Chapter 8: CREATING NEW ENVIRONMENTS.....	165
8-1: Introduction to Technology Editing.....	165
8-2: Converting between Technologies and Libraries.....	166
Converting Technologies to Libraries.....	166
Technology-Editing Mode.....	166
Converting Libraries to Technologies.....	166
Cleaning Up.....	167
Using Technology Libraries.....	167
8-3: Hierarchies of Technology Libraries.....	168
8-4: Miscellaneous Information.....	170
The Miscellaneous Information Cell.....	170
Additional Variables.....	171
8-5: The Layer Cells.....	172
8-6: Special Layer Information.....	176
8-7: The Arc Cells.....	177
Creating and Deleting Arc Cells.....	177
Editing Special Arc Information.....	178



Table of Contents

Editing Arc Geometry.....	179
8–8: The Node Cells.....	180
Creating and Deleting Node Cells.....	180
Editing Special Node Information.....	180
Editing Node Geometry.....	181
Special Node Considerations.....	182
8–9: How Technology Changes Affect Existing Libraries.....	184
Adding layers, adding arcs, adding nodes, adding general information.....	184
Deleting layers.....	184
Deleting nodes, deleting arcs.....	185
Deleting general information.....	185
Modifying layers.....	185
Modifying arcs, modifying nodes.....	185
Modifying general information.....	186
8–10: Examples of Use.....	187
Example: Modifying a Layer's Look.....	187
Example: Creating a New Node.....	188
Chapter 9: TOOLS.....	191
9–1: Introduction to Tools.....	191
9–2: Design–Rule Checking.....	193
Incremental DRC.....	193
Hierarchical DRC.....	194
DRC Rules.....	195
Dracula DRC.....	196
9–3: Electrical–Rule Checking.....	197
Well and Substrate Checking.....	197
Antenna Rule Checking.....	198
9–4: Simulation.....	199
Verilog.....	199
SPICE.....	200
SPICE and Verilog Primitives.....	203
SPICE Plotting.....	204
FashHenry.....	206
9–5: Routing.....	208
Auto Stitching.....	208
Mimic Stitching.....	209
Maze Routing.....	209
River Routing.....	210
9–6: Network Consistency Checking (NCC, or LVS).....	211
Network Comparison.....	211
Fine–Tuning.....	212
Disambiguation.....	213
9–7: PLA and ROM Generation.....	214
Introduction to PLAs.....	214
The nMOS PLA Generator.....	215
The CMOS PLA Generator.....	215
The ROM Generator.....	216
9–8: Pad Frame Generation.....	217



Table of Contents

9-9: Silicon Compiler.....	220
9-10: VHDL Compiler.....	223
9-11: Compaction.....	225
9-12: Logical Effort.....	226
Chapter 10: SIMULATION.....	229
10-1: Introduction to Simulation.....	229
10-2: Simulator Operation.....	230
The Waveform Window.....	230
Test Vectors.....	232
Clocks.....	232
Simulator Control.....	233
10-3: VHDL Interface (ALS).....	237
10-4: Behavioral Models (ALS).....	238
10-5: Simulation Concepts (ALS).....	240
10-6: The Gate Entity (ALS).....	242
The i: and o: Statements (Input and Output).....	242
Signal References in the i: Statement.....	242
Signal References in the o: Statement.....	243
The t: Statement (Time Delay).....	243
The Delta Timing Distribution of the t: Statement.....	244
The Linear Timing Distribution of the t: Statement.....	244
The Random Probability Function of the t: Statement.....	245
The Fanout Statement.....	245
The Load Statement.....	246
The Priority Statement.....	246
The Set Statement.....	246
10-7: The Function Entity (ALS).....	248
Declaring Input and Output Ports.....	248
Other Specifications.....	249
Example of Function Use.....	249
10-8: The Model Entity (ALS).....	250
The Set Statement.....	251
10-9: Documenting the Netlist (ALS).....	252
Chapter 11: INTERPRETERS.....	253
11-1: Introduction to Interpreters.....	253
11-2: The Lisp Interface.....	254
Session Control.....	254
Database Structure.....	254
Database Examination.....	255
Basic Synthesis.....	256
Hierarchy.....	259
Modification.....	260
Search.....	261
Views.....	263
Libraries.....	263
Technologies.....	263
Tools.....	264



Table of Contents

Miscellaneous.....	265
11-3: The TCL Interface.....	266
Session Control.....	266
Database Structure.....	266
Database Examination.....	267
Basic Synthesis.....	268
Hierarchy.....	270
Modification.....	272
Search.....	273
Views.....	274
Libraries.....	275
Technologies.....	275
Tools.....	275
Miscellaneous.....	276
11-4: The Java Interface.....	278
Session Control.....	278
Java used in Parameters.....	278
Database Structure.....	279
Database Examination.....	280
Basic Synthesis.....	281
Hierarchy.....	283
Modification.....	286
Search.....	287
Layers and Polygons.....	288
Views.....	289
Libraries.....	290
Technologies.....	290
Tools.....	290
Miscellaneous.....	291
11-5: Interpreter Attributes.....	293
Chapter 12: MENU SUMMARY.....	301
12-1: The File Menu.....	301
New Library... [3-9].....	301
Open Library... [3-9].....	301
Import [3-9].....	302
IO Options [3-9], [7-3].....	303
Close Library [3-9].....	306
Save Library [3-9].....	306
Save Library As... [3-9].....	306
Save All Libraries [3-9].....	306
Export [3-9].....	306
Change Current Library... [3-9].....	308
List Libraries [3-9].....	308
Rename Library... [3-9].....	308
Mark All Libraries for Saving [3-9].....	308
Print... [4-9].....	308
Print Options... [4-9].....	309
Quit.....	309



Table of Contents

12-2: The Edit Menu.....	310
New Cell Instance... [3-3].....	310
New Analog Part [7-6].....	311
New SPICE Part [9-4] [7-6].....	311
New Pure-Layer Node... [6-10] [7-1].....	312
New Special Object.....	312
New Node Options... [6-2].....	313
Cut [6-1] [4-10].....	314
Copy [6-1] [4-10].....	314
Paste [6-1] [4-10].....	314
Duplicate [6-1].....	315
Undo [6-7].....	315
Redo [6-7].....	315
Rotate [2-6].....	315
Mirror [2-6].....	315
Size [2-5].....	315
Move [2-4] [4-7] [6-5].....	316
Erase [2-3].....	316
Erase Geometry [2-3].....	316
Array... [6-4].....	317
Insert Jog in Arc [2-2].....	317
Change... [6-6].....	317
Cleanup Cell.....	318
Selection.....	318
Special Function.....	320
12-3: The Cells Menu.....	322
Edit Cell... [3-2].....	323
Cross-Library Copy... [3-10].....	324
Duplicate Current Cell [3-2].....	324
Project Management [6-11].....	324
Cell Options... [3-7].....	325
Cell Explorer... [3-7].....	326
Describe this Cell [3-7].....	326
General Cell Lists... [3-7].....	327
Special Cell Lists [3-7].....	327
Down Hierarchy [3-5].....	328
Down Hierarchy in Place [3-5].....	328
Up Hierarchy [3-5].....	328
Expand Cell Instances [3-4].....	328
Unexpand Cell Instances [3-4].....	329
Look Inside Highlighted [3-4].....	329
Package into Cell... [3-8].....	329
Extract Cell Instance [3-8].....	329
New Version of Current Cell [3-2].....	330
Delete Unused Old Versions [3-2].....	330
Read Text Cell... [4-10].....	330
Write Text Cell... [4-10].....	330
12-4: The Arc Menu.....	331
Rigid [5-3].....	331



Table of Contents

Non-Rigid [5-3].....	331
Fixed-angle [5-3].....	331
Not Fixed-angle [5-3].....	332
Negated [5-4].....	332
Directional [5-4].....	332
Ends-extend [5-4].....	332
Reverse [5-4].....	332
Skip Head [5-4].....	332
Skip Tail [5-4].....	332
New Arc Options... [5-5].....	333
Curve through Cursor [5-4].....	333
Curve about Cursor [5-4].....	333
Remove Curvature[5-4].....	333
12-5: The Export Menu.....	334
Create Export... [3-6].....	334
Re-Export Everything [3-6].....	335
Re-Export Highlighted [3-6].....	335
Re-Export Power and Ground [3-6].....	335
Delete Export [3-6].....	335
Delete All Exports on Highlighted [3-6].....	335
Delete All Exports in Area [3-6].....	335
Move Export [3-6].....	335
Rename Export... [3-6].....	336
Summarize Exports [3-6].....	336
List Exports [3-6].....	336
Show Exports [3-6].....	336
Port and Export Options... [3-6].....	336
Show Ports on Node [3-6].....	336
Add Exports from Library... [3-6].....	337
12-6: The View Menu.....	338
New View Type... [3-11].....	339
Delete View Type... [3-11].....	339
Change Cell's View... [3-11].....	339
Frame Options... [7-6].....	339
Icon Options... [3-12].....	340
Edit Layout View [3-11].....	340
Edit Schematic View [3-11].....	340
Edit Multi-Page Schematic View... [3-11].....	340
Edit Icon View [3-11].....	340
Edit VHDL View [3-11].....	340
Edit Documentation View [3-11].....	341
Edit Skeleton View [3-11].....	341
Edit Other View... [3-11].....	341
Make Layout View... [3-12].....	341
Make Schematic View [3-12].....	341
Make Multi-Page Schematic View... [3-11].....	341
Make Icon View [3-12].....	341
Make VHDL View [3-12].....	341
Make Documentation View [3-11].....	342



Table of Contents

Make Skeleton View [3-12].....	342
Make Other View... [3-11].....	342
12-7: The Windows Menu.....	343
Fill Window [4-4].....	343
Redisplay Window [4-1].....	344
Zoom Out [4-4].....	344
Zoom In [4-4].....	344
Special Zoom [4-4].....	344
Left [4-4].....	345
Right [4-4].....	345
Up [4-4].....	345
Down [4-4].....	345
Panning Distance [4-4].....	345
Center [4-4].....	345
Saved Views... [4-4].....	346
Toggle Grid [4-7].....	346
Grid Options... [4-7].....	346
Alignment Options... [4-7].....	346
New Window [4-3].....	347
Delete Window [4-3].....	347
Window Partitions [4-3].....	347
Adjust Position [4-3].....	347
Layer Visibility... [4-5].....	348
Color Options [4-6].....	348
Layer Display Options... [4-6].....	349
Text Options... [4-10], [6-8].....	350
3D Display [4-11].....	350
Component Menu... [4-8].....	351
Messages Window [4-2].....	351
12-8: The Info Menu.....	353
Get Info [2-4] [6-10] [2-5] [5-3] [3-6] [6-8].....	354
Attributes [6-8].....	356
List Networks [6-9].....	359
List Connections on Network [6-9].....	359
List Exports on Network [3-7] [6-9].....	359
List Exports below Network [3-7] [6-9].....	359
List Geometry on Network [6-9].....	359
List Layer Coverage [3-7].....	359
Rename Network... [6-9].....	360
Help... [1-10].....	360
See Manual.....	360
Tutorial [1-10].....	360
Option Control [6-3].....	360
Measure Distance [4-7].....	361
User Interface.....	361
Check and Repair Libraries [6-12].....	362
About Electric.....	362
12-9: The Technology Menu.....	363
Change Current Technology... [7-1].....	364



Table of Contents

Technology Options... [7-1].....	364
Change Units... [7-2].....	365
Document Technology [7-1].....	365
Describe Current Technology [7-1].....	365
Convert and Edit Technology... [8-2].....	365
Load Technology Library [8-2].....	366
Delete Technology... [8-2].....	366
Rename Technology... [8-2].....	366
Edit Primitive Node... [8-8].....	366
Edit Primitive Arc... [8-7].....	366
Edit Layer... [8-5].....	367
Edit Next Primitive [8-5] [8-7] [8-8].....	367
New Primitive [8-5] [8-7] [8-8].....	367
Reorder Primitives [8-5] [8-7] [8-8].....	367
Edit Colors... [8-6].....	367
Edit Design Rules... [8-6].....	367
Edit Variables... [8-4].....	368
Edit Library Dependencies... [8-3].....	368
Edit Miscellaneous Information [8-4].....	369
Identify Primitive Layers [8-7] [8-8].....	369
Identify Ports [8-8].....	369
Delete this Primitive [8-5] [8-7] [8-8].....	369
12-10: The Tools Menu.....	370
DRC [9-2].....	370
Simulation (Built-in) [10-1] [10-2].....	373
Simulation (SPICE) [9-4].....	375
Simulation (Verilog) [9-4].....	377
Simulation (Others) [9-4].....	379
Electrical Rules [9-3].....	381
Network [6-9] [9-6].....	382
Logical Effort [9-12].....	384
Routing [9-5].....	386
Generation.....	387
VHDL Compiler [9-10].....	388
Silicon Compiler [9-9].....	390
Compaction [9-11].....	391
List Tools [9-1].....	391
Language Interpreter [11-1].....	391





Using the Electric™ VLSI Design System

Steven M. Rubin

Version 7.00

July 14, 2004

Copyright (c) 2004 [Static Free Software](#).

Electric™ is a trademark of [Static Free Software](#).

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that they are labeled prominently as modified versions, that the authors' names and title from this version are unchanged (though subtitles and additional authors' names may be added), and that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.





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.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 1: INTRODUCTION



1-2: Requirements



Electric can run on all of the popular computer platforms in use today. Besides being UNIX compatible, it can also run under Windows and on the Macintosh. Separate installation instructions are given in the following sections.

Windows users must be running Windows 95, Windows NT 4.0, or any later version (98/ME/2000/XP).

Macintosh users must be running System 7 or later.

UNIX users will find that the system runs on most variants.

Electric is not a small program. It needs from 8 to 16 megabytes of memory, depending on the platform. However, any serious design effort demands much more. This is because the program grows with the amount of circuitry being designed, so large designs require large amounts of physical memory to avoid thrashing delays.

On disk, you will need at least 30 megabytes of free space in order to install Electric. This is because the source code is 15 megabytes, and there are various support files and libraries as well. Keep in mind that a single distribution includes support files for UNIX, Windows, and Machintosh systems, so you can eventually prune what you do not need.

In addition to memory requirements, the workstation must be color (Windows requires 65536 colors or more).

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 1: INTRODUCTION



1-3: UNIX Installation



Electric runs on most UNIX variants, including SunOS, Solaris, BSD, HPUX, AIX, and (of course) GNU/Linux.

The Electric distribution is a single file in UNIX "tar" format, GNU-zipped (see <http://www.gzip.org> for more information). To install Electric, follow this procedure:

- Extract the contents of the "tar" file. This will create a top-level directory called **electric-VERSION** with four subdirectories: **src**, **lib**, **examples**, and **html**.
- To configure for your system, go into the top-level directory and type:

./configure

This will examine the system and create a file called **Makefile**.

- To build electric you now only need to type **make**. This compiles Electric and creates the executable file **electric** in the top level.
 - Type **./electric** to run the system.
-

Here are some detail on the Electric distribution:

- The **src** directory contains the source code. It is hierarchically organized by function.
- The **html** directory contains one subdirectory, **manual**, which is this manual in HTML format. To see the document, point your browser to the file **index.html** inside of the **manual** directory.
- The **examples** directory has some demo files.
- Electric uses "widget libraries" to control the windows on the display. The default widget library is **Motif**. You can use **Lesstif**, but has some bugs (you will have to remove the "XtDestroyWidget()" call in "DiaDoneDialog()"). You can also use **Qt** by editing the **Makefile** after running "configure" (comments near the top explain what to do). Note that when you download these packages, you must download both the libraries for your system and the "devel" package which has header files for the compiler. Also note that many systems use shared libraries for these widget packages, and this may require some additional steps when installing. This is because the libraries get installed in a place that the shared library system doesn't know about. If you have superuser access, you can use **ldconfig** to tell the system where to find the libraries. Otherwise, you can use the **LD_LIBRARY_PATH** environment variable (on AIX use **LIBPATH** and on HP-UP use **SHLIB_PATH**). This variable is a colon-separated list of paths to be searched for shared libraries. For example, this setting will work on many systems:

```
LD_LIBRARY_PATH = /usr/X11R6/lib/
```



```
export LD_LIBRARY_PATH
```

- Electric's TrueType package is Rainer Menzner's "T1Lib". You can get it from here: <ftp://sunsite.unc.edu/pub/Linux/libs/graphics>

Once you have it, unpack it (it will create a directory with the version number as part of its name, for example "T1-1.1.0"), go into that directory, type `./configure` and then type `make`. You can then install with `make install`.

Now go to the Electric directory and edit the "Makefile". Near the top are comments labeled "T1LIB TRUETYPE". Uncomment the two lines (remove the "#" from the beginning of the lines) and change the lines so that they point into the T1Lib folder that you have just installed. For example, if the T1Lib libraries installed into `/usr/local/lib` and the T1Lib headers installed into `/usr/local/include`, then you want these lines to read:

```
TRUETYPE_LIBS = /usr/local/lib/libt1.a
TRUETYPE_FLAGS = -DTRUETYPE=1 -I/usr/local/include
```

Next (this is the most painful part), you have to set the environment variable `T1LIB_CONFIG` to point to the file `t1lib.config`. This file may be installed in `/usr/local/share/t1lib`. There will certainly be a copy in the "examples" folder of the distribution. The catch here is that this file has relative path names in it which must be converted to absolute. So, if you have extracted the T1Lib distribution into the folder `/home/strubin/T1-1.1.0`, then the file should look like this:

```
This is a configuration file for t1lib
FONTDATABASE=/home/strubin/T1-1.1.0/examples/FontDataBase
ENCODING=/home/strubin/T1-1.1.0/Fonts/enc:.
AFM=/home/strubin/T1-1.1.0/Fonts/afm:.
TYPE1=/home/strubin/T1-1.1.0/Fonts/type1:.
```

Finally, rebuild Electric with the TrueType library. When you run it, you will get a warning if any of the TrueType initialization fails (in which case it will revert to the non-TrueType code). Otherwise, you have it. Note that Electric uses the first font in the database by default. To change the font that Electric uses, set the environment variable `ELECTRIC_TRUETYPE_FONT` to the desired font name. You can see a list of available fonts by setting this environment variable to an unknown name, in which case Electric will show all fonts in its error message.

- On Solaris, when using the Forte C compiler, uncomment the line that starts with `FORTECFLAGS` in **Makefile**.
- Installation of Electric requires some care. Although Electric will run properly from the directory where it was built, you cannot move the executable to a different location and expect it to work. This is because Electric makes use of a collection of *support files*. The main support file is called **.cadrc**. In addition, Electric needs to find the **lib** and **html** directories. If these support files cannot be found, Electric will not be able to initialize its graphical user interface (just type `--quit` to exit the program if this happens).

The **make install** command will place the executable and the support files in a public location, but they may not be together. For example, it is not uncommon for the executable to be placed in `/usr/local/bin`, but the support files in `/usr/local/lib/electric`. Because of this, the executable needs to know where the support files are located. There are three ways to do this:

- ◆ You can set the `ELECTRIC_LIBDIR` environment variable to point to the location of the support files.
- ◆ You can change the `#define` of `"LIBDIR"` in `"src/include/config.h"` to point to the location of the support files.
- ◆ You can keep a local copy of `".cadrc"` (this file can be in your home directory or in the current directory). Inside of the `".cadrc"` file, change the `"electric library default-path"` command to point to the remaining support files (the **lib** and **html** directories).

You can use the command **make install.html** to install the online manual in a public place (typically



"/usr/local/share/doc/electric/html"). Be sure that the #define of "DOCDIR" in "src/include/config.h" agrees with this path, or else the **See Manual** command will not work.

- The IRSIM simulator and LISP interpreter are not distributed as part of the GNU download. Users who wish to add these facilities must acquire the Static Free Software extras [described here](#). Use the same procedure to extract this extension file (which will also be a GNU-zipped tar file). Extract it into the same location as the main source distribution and it will add the necessary files to the source tree. Then edit **Makefile** and you will find the instructions necessary to enable these facilities. It will be necessary to recompile all of Electric when adding these extras.

- To add Java, follow these instructions:

- ◆ Download the Java Development Kit (JDK) from <http://java.sun.com>. Install it. For the purposes of these instructions, assume that it is installed into `/usr/java/jdk`. If you install it elsewhere, adjust these instructions accordingly.
- ◆ After configuration, but before making Electric, edit the **Makefile** and uncomment the lines near the top that enable Java. Change the definition of LANGJAVA_DIR to point to the installed JDK location.

- ◆ On Solaris, add this string to the environment variable LD_LIBRARY_PATH:

```
:/usr/java/jdk/jre/lib/sparc:/usr/java/jdk/jre/lib/sparc/classic
```

- ◆ On GNU/Linux, add this string to the environment variable LD_LIBRARY_PATH:

```
:/usr/java/jdk/jre/lib/i386:/usr/java/jdk/jre/lib/i386/classic:/
```

- ◆ Be sure to export "LD_LIBRARY_PATH" if your shell requires it.

- ◆ Electric's Java interface works better if you also install the "Bean Shell" (see www.beanshell.org). Download the ".jar" file and place it in the "java" subdirectory of your "lib" directory. You must use version 1.1Alpha4 or later (version 1.01 is not acceptable).

- To add the TCL interpreter, download it from <http://www.tcl.tk> and install it. Then edit **Makefile** and you will find the instructions necessary to enable the interpreter.

- Electric has two ways to control the display. By default, the system runs on any depth monitor, but is slow on older machines and must be run locally (that is, the client and the server must be on the same computer). The alternate method of display is faster and can run over the network, but it can only support displays that are set to 8bpp (8 bits per pixel). In addition, this alternate method will suffer from "colormap flashing" when the cursor enters and leaves the Electric windows. To switch to this alternate method, edit **Makefile** after running "configure" (comments near the top explain what to do). Note also that Motif and Lesstif do not work well with this alternate display method, so you will also have to switch to using the Athena widgets.

- Electric is able to invoke the SPICE simulator automatically. In order to do this, it needs to know the location of this program. You can change the #define of "SPICELOC" in **src/include/config.h**, or you can set the environment variable ELECTRIC_SPICELOC.

- If you wish the "file" command to recognize Electric libraries, add these two lines to "/etc/magic" (or wherever the "magic" information is stored):

```
0 long 031176377777 Electric library
0 long 037777774711 Electric library
```

- There are two command-line arguments that can be given which will control the display. If you use the "-m" option, Electric will look for multiple displays and use them (it searches for files named "/dev/fb*"). If you use the "-geom WxH+X+Y", it will set the graphics window to be "W" wide, "H" high, and with its corner at (X, Y).

- Additional X-Windows options can be typed into the file ".Xdefaults". The resources "Electric.font0" through "Electric.font8" set the font to use for point sizes 4, 6, 8, 10, 12, 14, 16, 18, and 20. The resource "Electric.fontmenu" controls the text used in the component menu, and the resource "Electric.fontedit" controls the text used in the text editor. Here is a sample line from the file:

```
Electric.font5: -misc-fixed-medium-r-normal-*-140-*-*-*-*
```

To see what all of these fonts look like, load the library **samples.txt** (with the **Readable Dump** subcommand of the **Import** command of the **File** menu) and edit the cell **tech-Artwork**. The



top part of the cell shows text in sizes 4 through 20.

Don't forget to restart X after making changes to the ".Xdefaults" file.

- Electric can speak your language! Currently, it has been translated into French. Contact Static Free Software if you are interested in doing a translation. To use this facility, edit the "Makefile" and follow the instructions for "Internationalization". You must then set the environment variable "LANGUAGE" to the proper language ("fr" for French). On Solaris, you must also set the environment variable "NLSPATH" to point to Electric's "lib/international" directory. At any time, you can disable the foreign language and return to English by moving the translation files. These files are in the "lib/international" folder, with a subfolder that has the language name (for example, French translations are in "lib/international/fr"). Beneath that is a folder called "LC_MESSAGES" and inside of that are the translation files.
-

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 1: INTRODUCTION



1-4: Macintosh Installation



Macintosh users must run System 7 or later. Electric comes with project files for Metrowerks, although it has been built with MPW and THINK_C. System 10 users can choose between Qt (which costs money) and ProjectBuilder (which is not fully working yet).

The Electric distribution is a single file in UNIX "tar" format, GNU-zipped. On the Macintosh, use the program [MacGzip](#) to un-zip the file, and [MacTar](#) to un-tar the file. To install Electric, follow this procedure:

- After the files have been extracted, there will be a top-level directory called **electric-VERSION** with four subdirectories: **src**, **lib**, **examples**, and **html**.
- Beware of Macintosh line-feed conventions, which are different from those on other operating systems. If you use an older "tar" program (other than "MacTar"), you may need to set the "Convert Newlines" option before extracting the "tar" file. Also, if you use "Internet Config", check to be sure its "Change Newline" setting set. To be sure that the extraction has worked properly, examine the file **cadrc**, which is in the top level directory. This file should have less than 10 lines of text. If the file appears as a single line, or if there are spurious unprintable characters at the start or end of each line, then the text conversion has been done incorrectly.
- For System 7, 8, and 9, there is a Metrowerks project (called **Electric.xml**). Run Metrowerks, import this file, and save it in the top level, alongside the **src** directory. Due to the size of the code that is being built, you may have to increase the size of the Metrowerks partition.
- For System 10, there are two ways to go: *Qt* or *ProjectBuilder*. Qt is the only fully-working solution, but unfortunately it is not free on the Macintosh (it is actually quite expensive). Also, you need Qt release 3.1.0 or later. To build with Qt, use a terminal window and type **./configure** to generate a **Makefile**. Edit the **Makefile** and switch to Qt widgets (uncomment the Qt part, comment the Motif part, and in the Qt section, change comments to switch to "Qt on Macintosh"). Qt also requires that you set "Font Smoothing Style" to "Standard" in the "General" System Preferences. If you do not have Qt, then there are ProjectBuilder files (called **Electric.pbproj** and the **English.lproj** folder). Be warned that this is not fully debugged, so use with care.
- Compile Electric. This will create the application **Electric**.
- Double-click the **Electric** application to run the system.

Here are some detail on the Electric distribution:

- The **src** directory contains the source code. It is hierarchically organized by function.
- The **html** directory contains one subdirectory, **manual**, which is this manual in HTML format. To see the document, point your browser to the file **index.html** inside of the **manual** directory.
- The **examples** directory has some demo files.



- The IRSIM simulator, LISP interpreter, and Foreign language interfaces are not distributed as part of the GNU download. Users who wish to add these facilities must acquire the Static Free Software extras [described here](#). Use the same procedure to extract this extension file (which will also be a GNU-zipped tar file). Extract it into the same location as the main source distribution and it will add the necessary files to the source tree. Then import the project file **ElectricSFS.xml** to create the Metrowerks project for Electric with the language extension.
- To add the TCL interpreter, follow these instructions:
 - ◆ Download ActiveTcl from <http://www.tcl.tk> and install it.
 - ◆ If using Qt/System 10, edit **Makefile** and add TCL. Otherwise:
 - ◇ In the compiler, add an include path to the installed TCL "include" directory.
 - ◇ Also in the compiler, add the appropriate TCL library to the project.
 - ◆ Edit the appropriate "mac" include file in **src/include** (for example, **macfsheaders.h**) and uncomment the definition of "FORCETCL".
- To add a Java interpreter (System 10 only) follow these instructions:
 - ◆ Download Java from <http://java.sun.com> and install it.
 - ◆ If using Qt, edit **Makefile** and add Java. Otherwise:
 - ◇ In the compiler, add an include path to the installed Java "include" directory.
 - ◇ Also in the compiler, add the appropriate Java library to the project.
 - ◆ Edit the appropriate "mac" include file in **src/include** (for example, **macfsheaders.h**) and uncomment the definition of "FORCEJAVA".
- Installation of Electric requires some care. Although Electric will run properly from the directory where it was built, you cannot move the executable to a different location and expect it to work. This is because Electric makes use of a collection of *support files*. The main support file is called **cadrc**. In addition, Electric needs to find the **lib** and **html** directories. If these support files cannot be found, Electric will not be able to initialize its graphical user interface (just type "-quit" to exit the program if this happens). It is sufficient to move the support files, along with the executable, to a public location. Then make an alias to the executable and place that anywhere you like. When the alias is run, the directory with the executable will become the current directory, and all of the needed support files will be found.
- Electric can speak your language! Currently, it has been translated into French. Contact Static Free Software if you are interested in doing a translation. To use this facility, you must obtain the Static Free Software extras and build the "International" version of Electric in **ElectricIntl.xml** or **ElectricSFSIntl.xml**. Before compiling, set the desired language by changing the routine "elanguage()" in "graph/graphmac.c".
At any time, you can disable the foreign language and return to English by moving the translation files. These files are in the "lib/international" folder, with a subfolder that has the language name (for example, French translations are in "lib/international/fr"). Beneath that is a folder called "LC_MESSAGES" and inside of that are the translation files.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 1: INTRODUCTION



1–5: Windows Installation



Electric runs under Windows 95/98/ME, Windows NT 4.0, Windows 2000, or Windows XP. The system compiles with Visual C++ 5.0 or later (project files are included).

The Electric distribution is a single file in UNIX "tar" format, GNU-zipped. This can be extracted by a number of programs, including "WinZip" (see <http://www.gzip.org> for more information). To install Electric, follow this procedure:

- Extract the contents of the "tar" file. When using WinZip, make sure that the "TAR File Smart CR/LF Conversion" box is checked in the "Configuration..." dialog of the "Options" menu. Once extracted, you will have a top-level directory called **electric-VERSION** with four subdirectories: **src**, **lib**, **examples**, and **html**.
- For users of Visual C++ 5.0 or 6.0, open the workspace file **Electric.dsw** (both it and the associated file **Electric.dsp** are in the top level, alongside the **src** directory). Visual Studio .NET users can open **Electric.vcproj**. If you have trouble with any of these files, use the MAKE file **Electric.mak**. Compile Electric. This will create a new directory in the top level called **Debug**, which will contain all of the object files.
- Inside of the **Debug** directory, you will find the executable file **Electric**. Move this file out of the **Debug** directory and place it in the top-level directory. Double-click the **Electric** executable to run the system.

Here are some detail on the Electric distribution:

- The **src** directory contains the source code. It is hierarchically organized by function.
- The **html** directory contains one subdirectory, **manual**, which is this manual in HTML format. To see the document, point your browser to the file **index.html** inside of the **manual** directory.
- The **examples** directory has some demo files.
- The IRSIM simulator, LISP interpreter, and Foreign language interfaces are not distributed as part of the GNU download. Users who wish to add this facility must acquire the sources [separately](#). Use the same procedure to extract this extension file (which will also be a GNU-zipped tar file). Extract it into the same location as the main source distribution and it will add the necessary files to the source tree. Then use the file **ElectricLang.dsw** to build Electric with the language extension. The resulting executable will be in the **DebugLang** directory.
- To add the Java interpreter, follow these instructions:
 - ◆ Download the Java Development Kit (JDK) from <http://java.sun.com>. Install it. Although it can be placed anywhere, these instructions will assume that you have installed it in location **C:\Program Files\JavaSDK**. If you install it elsewhere, adjust these instructions accordingly.



- ◆ Edit the environment variables in the "System" Control Panel. On some systems, you click on the "Environment" tab; on others, click on the "Advanced" tab and then click the "Environment Variables" button. Under "System variables", select "Path" and in the "Value:" area, add this string to the end:

```
;C:\Program Files\JavaSDK\jre\bin\classic;C:\Program
Files\JavaSDK\bin
```

On some newer versions of the Java Development Kit, you may also have to include this path:

```
;C:\Program Files\JavaSDK\bin\client
```

On Windows 95 and Windows 98 systems, you may have to edit C:\AUTOEXE.BAT and append this to the PATH variable. You must restart your computer after making this change.

- ◆ In Visual C++ 5.0 or 6.0, use the "Settings" command of the "Project" menu. Select the "C/C++" tab and the "Preprocessor" category. In the "Preprocessor definitions" area, add this to the end:

```
,FORCEJAVA=1
```

In the "Additional include directories" area, add this to the end:

```
;C:\Program Files\JavaSDK\include,C:\Program
Files\JavaSDK\include\win32
```

Select the "Link" tab and the "General" category. In the "Object/library modules" area, enter this:

```
jvm.lib
```

Select the "Link" tab and the "Input" category. In the "Additional library path" area, enter this:

```
C:\Program Files\JavaSDK\lib
```

- ◆ In Visual Studio .NET, right-click on the "Electric" solution and choose "Properties". Select "C/C++" on the left and choose the "General" category under it. In the "Additional Include Directories" area, add this to the end:

```
;C:\Program Files\JavaSDK\include,C:\Program
Files\JavaSDK\include\win32
```

Next choose the "Preprocessor" category of "C/C++" and in the "Preprocessor Definitions" area add this to the end:

```
;FORCEJAVA=1
```

Select "Linker" on the left and choose the "General" category under it. In the "Additional Library Directories" area, enter this:

```
;C:\Program Files\JavaSDK\lib
```

Next choose the "Input" category of "Linker" and in the "Additional Dependencies" area enter this:

```
jvm.lib
```

- ◆ Electric's Java interface works better if you also install the "Bean Shell" (see www.beanshell.org). Download the ".jar" file and place it in the "java" subdirectory of your "lib" directory. You must use version 1.1Alpha4 or later (version 1.01 is not acceptable).
- ◆ Once Java is installed, you must compile the ROM generator. In a command window, change directories to the **lib\java** directory and run the command:


```
javac romgen.java
```
- To add the TCL interpreter, follow these instructions:
 - ◆ Download ActiveTcl from <http://www.tcl.tk> and install it.
 - ◆ In the compiler, edit the Project Settings and find the field "Additional include directories" (under "C/C++"). Add a new path to the installed TCL Includes (typically "C:\Program Files\Tcl\include").
 - ◆ Also in the compiler, edit the Project Settings and find the field "Additional library path" (under "Linker"). Add a new path to the installed TCL Libraries (typically "C:\Program Files\Tcl\lib").



- ◆ Edit the file **src/include/config.h** and make sure that the constant "TCLLIBDIR" points to the proper location of the initialization files ("init.tcl" and others). This is typically "C:\Program Files\Tcl\lib\tcl8.3" (note that each backslash is doubled in this file, and you should follow this convention).
 - Electric must run on a display that is set to "65536 Colors" or "True Color". Anything less will cause the colors to appear wrong.
 - If you have trouble reading the cursor or icon files (".cur" or ".ico") you can find a text–encoded version of these binary files in **\src\graph\graphpc.uue**. Use "WinZip" to extract the files into the same directory.
 - Installation of Electric requires some care. Although Electric will run properly from the directory where it was built, you cannot move the executable to a different location and expect it to work. This is because Electric makes use of a collection of *support files*. The main support file is called **cadrc**. In addition, Electric needs to find the **lib** and **html** directories. If these support files cannot be found, Electric will not be able to initialize its graphical user interface (just type "--quit" to exit the program if this happens). It is sufficient to move the support files, along with the executable, to a public location. Then make a shortcut to the executable and place that anywhere you like. When the shortcut is run, the directory with the executable will become the current directory, and all of the needed support files will be found.
 - Electric can speak your language! Currently, it has been translated into French. Contact Static Free Software if you are interested in doing a translation. To use this facility, you must obtain the Static Free Software extensions and build the "International" version of Electric. Before compiling, set the desired language by changing the routine "elanguage()" in "graph/graphpccode.cpp". At any time, you can disable the foreign language and return to English by moving the translation files. These files are in the "lib/international" folder, with a subfolder that has the language name (for example, French translations are in "lib/international/fr"). Beneath that is a folder called "LC_MESSAGES" and inside of that are the translation files.
-

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



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 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 Integrated Circuit 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:

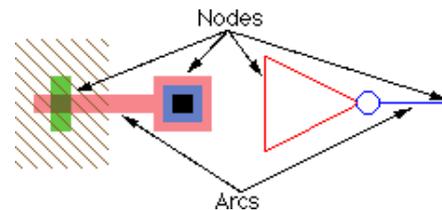
- **No node extraction.** Node extraction is not a separate, error–prone step. Instead, the connectivity is part of the layout description and is instantly available. This speeds up all network–oriented operations, including simulation, LVS, and electrical rules checkers.
- **No geometry errors.** Complex 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, the typical design iteration is to get the layout to be design–rule clean before comparing it to the schematics (LVS) because the extractor cannot run if the design rules are wrong. Then, when 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.

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 cell view 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 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. By declaring *exports* inside of the cell definition, their instances properly interface at



higher levels of the hierarchy.

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.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



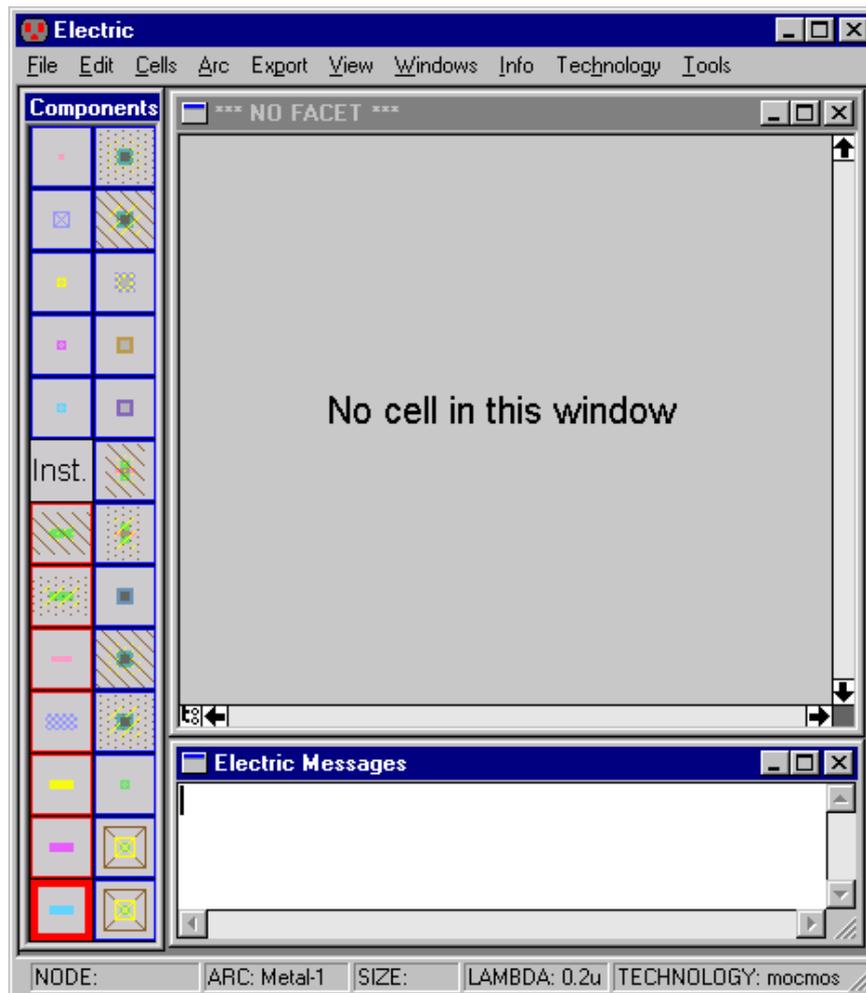
Chapter i: INTRODUCTION



1-7: The Display



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



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 *messages window* is a text window (typically at the bottom of the screen) which is used for all textual communication.



There is a *pull-down menu* along the top with command options. On some operating systems, the pull-down menu is part of the edit window, and on others it is separate.

The *components menu* is a palette that typically runs down the left side of the screen. It shows a list of nodes (blue outline) and arcs (red outline) that can be used in design. The current arc is highlighted with a bolder red outline.

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.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 1: INTRODUCTION



1–8: The Mouse



Electric's mouse button commands adapt to the different types of mice in the world:

Command	UNIX Three-Button	Windows Two-Button	Macintosh One-Button
Selection	left	left	click
Toggle ¹ Select	shift left	shift left	shift click
Select Another ²	ctrl left	ctrl left	ctrl click
Toggle Select Another	shift ctrl left	shift ctrl left	shift ctrl click
Special ³ Select	meta left	alt left	opt click
Toggle Special Select	shift meta left	shift alt left	shift opt click
Special Select Another	ctrl meta left	ctrl alt left	ctrl opt click
Toggle Special Select Another	shift ctrl meta left	shift ctrl alt left	shift ctrl opt click
Information	DOUBLE left	DOUBLE left	DOUBLE click
Creation / Tech. Edit	right	right	cmd click
Rectangle Select ⁴	meta right	alt right	cmd opt click
Rectangle Zoom ⁵	shift meta right	shift alt right	shift cmd opt click
Wire ⁶	shift right	shift right	shift cmd click

Electric presumes that Macintosh mice have 1–button, PC mice have 2, and UNIX mice have 2 or 3 (only 2 are used). Therefore, different versions of the mouse commands combine key presses with clicks.



For example, the *selection* button is used to highlight objects on the display. On systems with 1-button mice (i.e. Macintosh), the selection button is a plain click of the button, but on systems with two or three buttons, the selection button is the left button.

A note to users of the KDE 2.1 window system (Linux/UNIX): The Alt-Left button is used by the window system, and as a result, you may not be able to issue the Special Select button. To disable the window systems use of this button combination, use the KDE control tool, select "Look Feel", "Window behavior", "Actions" tab. Near the bottom, select "nothing" instead of "default move".

Notes from mouse table:

- 1: "Toggle" implies inversion of the selection, deselecting what is already selected and adding unselected objects.
 - 2: "Another" implies cycling through a list of objects that are under the cursor.
 - 3: "Special" allows the selection of "Hard-to-Select" objects.
 - 4: Defines a rectangular area.
 - 5: Defines a rectangular area and zooms into it.
 - 6: Same as "Creation" except that it does not connect if over another object. Also used in Technology Editing to make changes.
-

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 1: INTRODUCTION



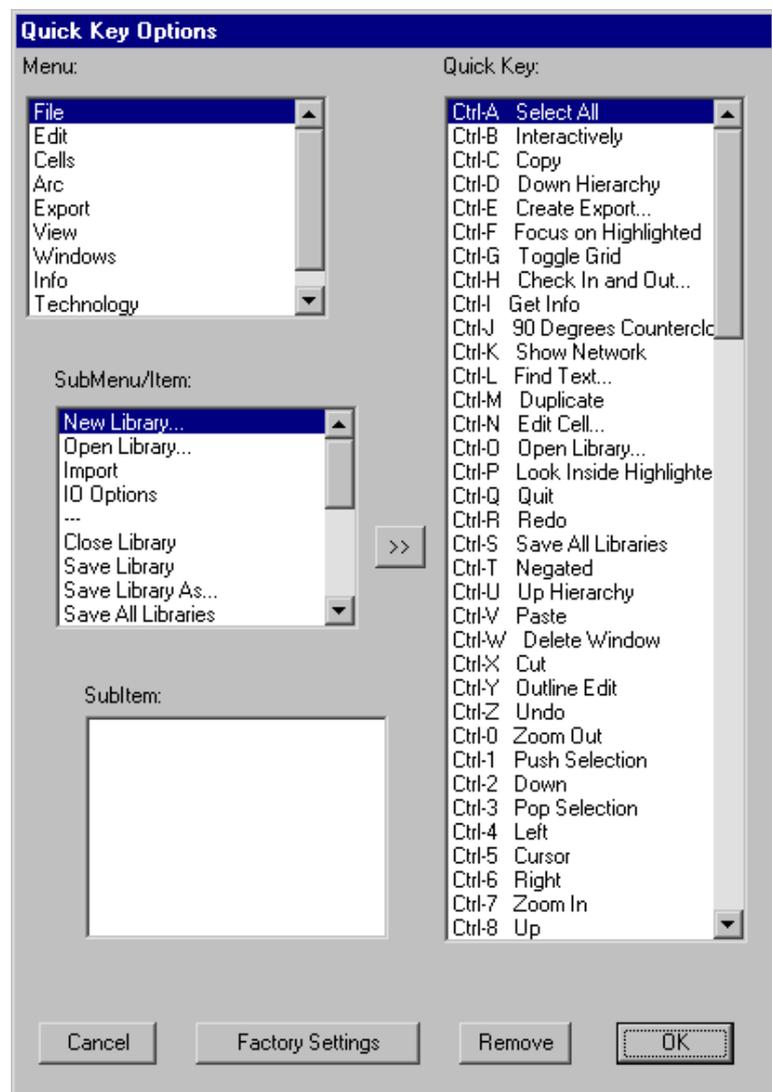
1-9: The Keyboard



Quick Keys

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 **Negated** command of the **Arc** menu has the quick key "Control-T". On the Macintosh, the menu shows "⌘T", indicating that you must hold the command key while typing the "T"; on Windows and UNIX systems, the menu shows "Ctrl-T", indicating that you must hold the Control key while typing "T". There are also unshifted quick keys (for example, typing the letter "A" repeats the last command by invoking the **Repeat Last Command** subcommand of the **User Interface** command of the **Info** menu).

To change the bindings of quick keys, use the **Quick Key Options...** subcommand of the **User Interface** command of the **Info** menu. The dialog shows the hierarchical structure of the pulldown menus on the left, and a list of quick keys on the right.



You can remove a quick key binding with the "Remove" button, and you can add a quick key binding with the ">>" button. The "Factory Settings" button restores default quick key bindings. Use the **Quick Key Options...** command with caution, because it customizes your user interface, making it more difficult for



other users to work at your station.

Note that the Function keys of most keyboards (F1 through F12) can also be attached to menu items. Be warned that many window systems take control of the higher function keys, and so attaching Electric commands to them may not work.

On Windows and UNIX systems, 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.

The Interrupt Key

Many operations take a long time in Electric. If you grow impatient, type the interrupt key. This key varies with the different platforms:

- On UNIX, type Ctrl-C (hold the Control key and type "c").
- On Windows, type  C (hold the Windows key and type "c").
- On a Macintosh, type  (hold the Command key and type ".").

Note that on UNIX systems, you have to type Ctrl-C in the messages window, because typing it in the edit windows invokes a menu item.

Also, if you have pressed the mouse button but not released it yet, you will note that many operations track the effects of your intended operation on the screen. To abort this operation, type the interrupt key or the letter "a".

Don't Type This Key

Electric has a highly complex command system that resides "under the surface." These commands can be invoked by typing the "-" key. However, because these commands are not intended for general use, do not type this key. If you do, a dialog will appear that will request a full command. Simply click the "Cancel" button to abort the dialog. Refer to the "Electric Internals Manual" for an explanation of the textual command language.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 1: INTRODUCTION

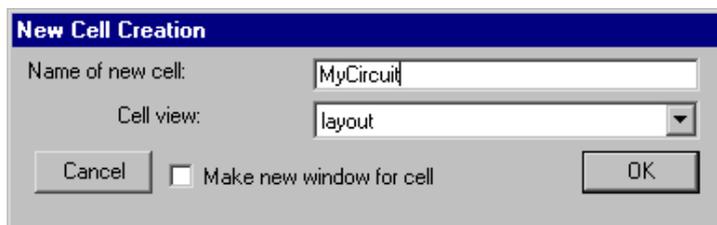
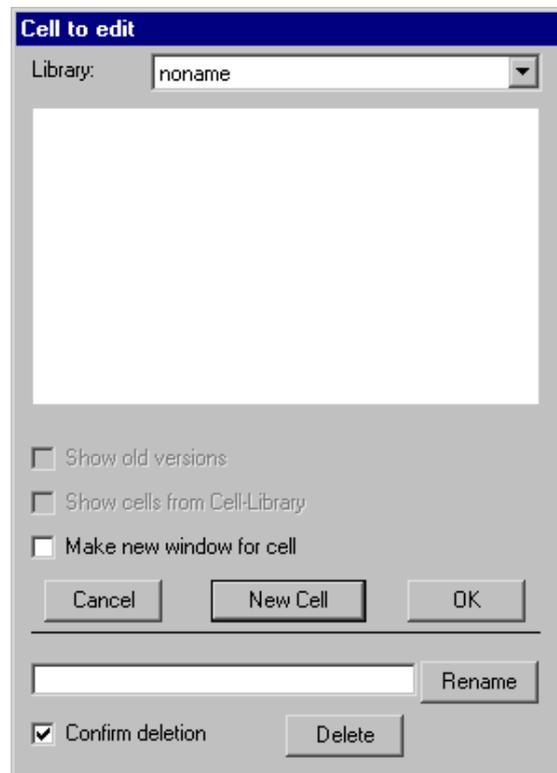


1-10: IC Layout Example



This section takes you through the design of some simple IC layout. The instructions here are also available by using the **Tutorial** command of the **Info** menu.

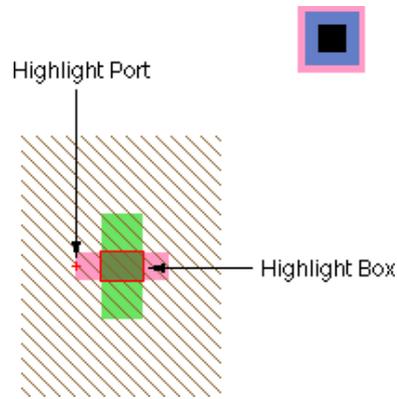
Before you can place any IC layout, the editing window must have a cell in it. Use the **Edit Cell...** command in the **Cells** menu. This will show a dialog with a list of existing cells (which is empty, because none exist yet). Click on the "New Cell" button at the bottom of this dialog to create a new cell. You will then see a dialog which asks you for information about this new cell.



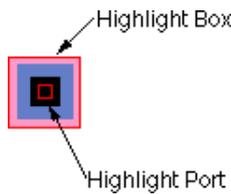
Type the name ("MyCircuit" is used here) and click OK. The editing window will no longer have the "No cell in this window" message, and circuitry may now be created.



Layout is placed by selecting nodes from the 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 sixth entry from the bottom in the right column of the component menu). The node on the bottom is called N-Transistor (sixth entry from the top in the right column of the component menu). Both of these nodes are from the MOSIS CMOS technology (which is listed as "mocmos" in the status area).



A *highlighted* node has two selected areas: the node and a port on that node. Note that the transistor is highlighted in the example above, and the contact is highlighted in the example here. The larger selected area is the node, and it surrounds the "important" part of the node (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).

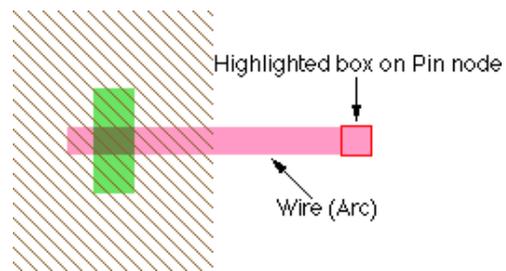
To highlight a node, use the *selection* button. The node, and the closest port to the cursor, will be selected. After highlighting, you can hold the mouse button down and drag the highlighted object to a new location. If nothing is under the cursor when the selection button is pushed, you may drag the 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 *toggle select* button. The *toggle select* 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.



To wire a component, select it, move the cursor away from the component, and use the *creation* 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.



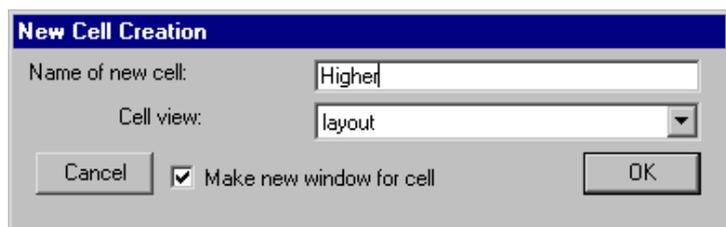
Once a wire has been created, the other end is highlighted (see above). This is the highlighting of a *pin* node that was created to hold the other end of the arc. Because it is a node, the *creation* button can be used again to continue the wire to a new location. If the creation command terminates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

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

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* and *slidable*, so the letters "FS" are shown when the wire is highlighted.

Select a wire and issue the **Rigid** command of the **Arc** menu. The letters change to "R" on the arc and the wire no longer stretches when components move. Find another arc and issue the **Not Fixed-angle** command in the **Arc** menu. 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.

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 **Edit Cell...** command of the **Cells** menu. Click on the "New Cell" button at the bottom, and make sure the "Make new window for cell" option is checked in the dialog. Then type the new cell name ("Higher" is used in the example here) and set its view.

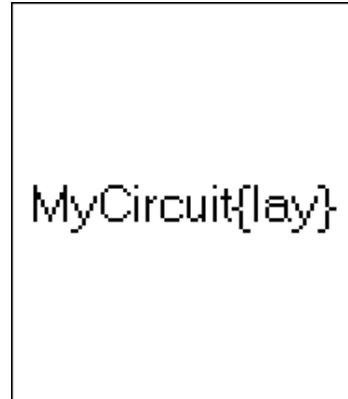


A new (empty) cell will appear in a separate window. Try creating a few simple nodes in this new window (place a contact or two).



Now place an instance of the other cell by using the **New Cell Instance...** command from the **Edit** 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.

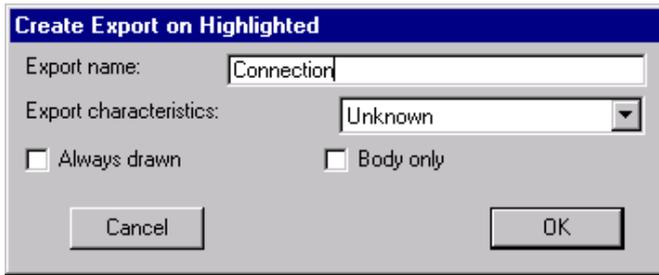
The box that appears is a node in the same sense as the contacts and transistors: it can be moved, wired, and so on. In addition, because the node contains subcomponents, you can see its contents by selecting it and using the **One Level Down** subcommand of the **Expand Cell Instances** command in the **Cells** menu. Note that if the objects in a cell no longer fit in the display window, use the **Fill Window** command from the **Windows** menu.



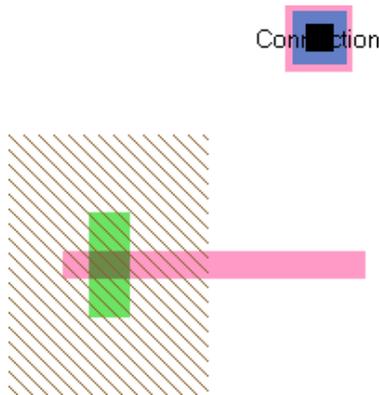
Before you can attach wires to the instance node, there must be connection sites, or *ports* on that node. *Primitive* nodes such as contacts and transistors already have their ports established, but you must explicitly create ports for cell instances.

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 from the **Export** menu. You will be





prompted for an export name and its characteristics (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.



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. Also, the **Help...** command from the **Info** menu is very useful. It displays a dialog with a list of subjects and offers information about each one.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 1: INTRODUCTION



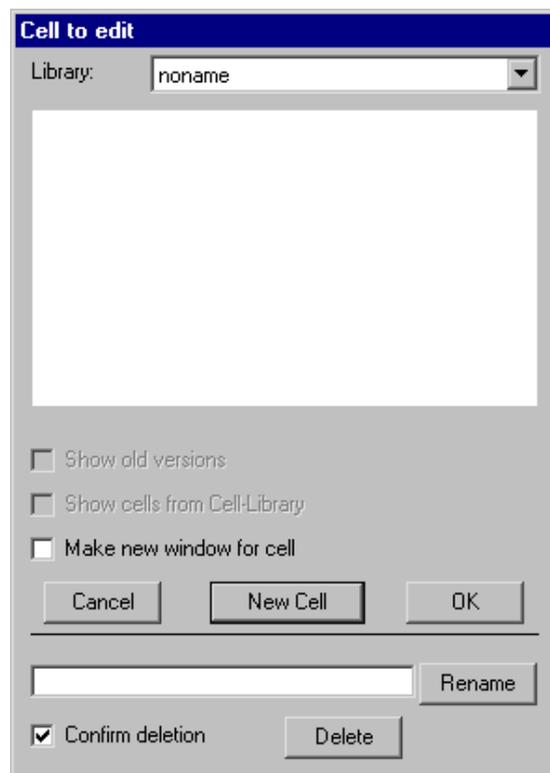
1-11: Schematics Example

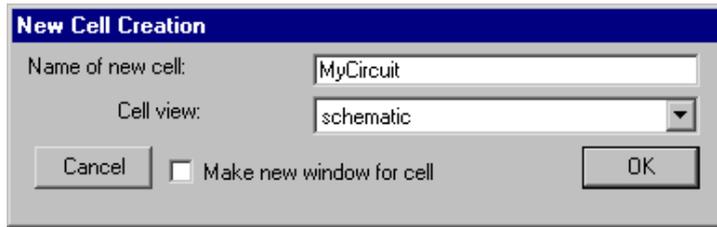


This section takes you through the design of some simple schematics.

Electric starts in IC design mode (MOSIS CMOS layout), so you have to switch to schematics before doing any design. Use the **Change Current Technology...** command from the **Technology** menu. Select the "schematic, digital" entry (you will have to scroll down to find it) and click OK. The symbols in the component menu on the left will change to a Digital schematics set. Analog components can be created by selecting the "schematic, analog" technology, but they may also be selected with the **New Analog Part** command in the **Edit** menu.

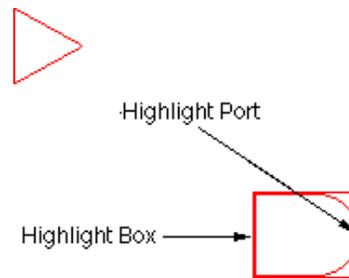
Before you can place any schematics, the editing window must have a cell in it. Use the **Edit Cell...** command in the **Cells** menu. This will show a dialog with a list of existing cells (which is empty, because none exist yet). Click on the "New Cell" button at the bottom of this dialog to create a new cell. You will then see a dialog which asks you for information about this new cell.



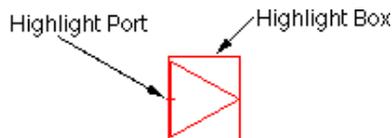


Type the name ("MyCircuit" is used here) and click OK. The editing window will no longer have the "No cell in this window" message, and circuitry may now be created.

Logic elements are placed by selecting nodes from the 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 a Buffer (found in the seventh entry from the bottom of the component menu). The node on the bottom is called an And (eighth entry from the bottom of the component menu). Both of these nodes are from the Schematic technology (which is listed as "schematics" in the status area).



A *highlighted* node has two selected parts: the node and a port on that node. Note that the And is highlighted in the example above, 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).

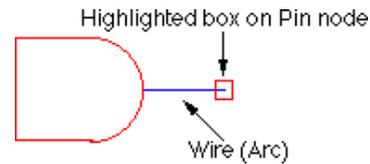
To highlight a node, use the *selection* button. The node, and the closest port to the cursor, will be selected. After highlighting, you can hold the mouse button down and drag the highlighted object to a new location. If nothing is under the cursor when the selection button is pushed, you may drag the 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 *toggle select* button. The *toggle select* 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.



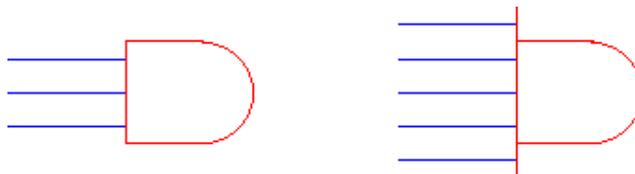
To wire a component, select it, move the cursor away from the component, and use the *creation* button. If you click the *creation* button and hold it without releasing, then you can move around and see where the wire will go when you do release.



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 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 *creation* button can be used again to continue the wire to a new location. If the creation command terminates over an existing component, the wire will attach to that component.

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 *creation* button to draw wires out of it. After three wires have been connected to the input side, the gate grows to accommodate 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, you must negate the wire connected to that gate. Select a wire and use the **Negated** command from the **Arc** menu. With this facility, you can construct arbitrary gate configurations.



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

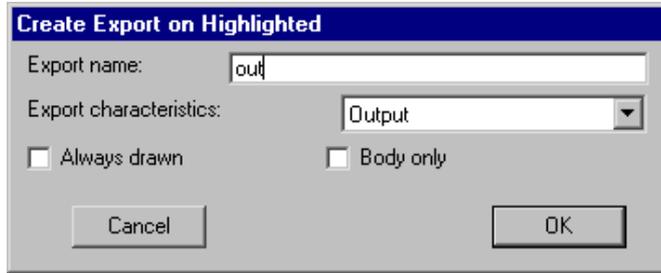
Once components are wired, moving them will also move their connecting wires. Notice that the wires stretch and move to maintain the connections. What actually happens is that the programmable constraint system follows instructions stored on the wires, and reacts to component changes. The default wire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

Select a wire and issue the **Rigid** command of the **Arc** menu. The letters change to "R" on the arc and the wire no longer stretches when components move. Find another arc and issue the **Not Fixed-angle** command in the **Arc** menu. 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.

Electric supports hierarchy by allowing you to create icons for a schematic and place them in another cell.

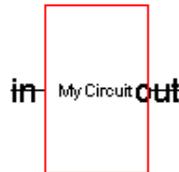
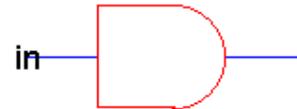


Before creating an icon, all connection points to the schematic should be defined.



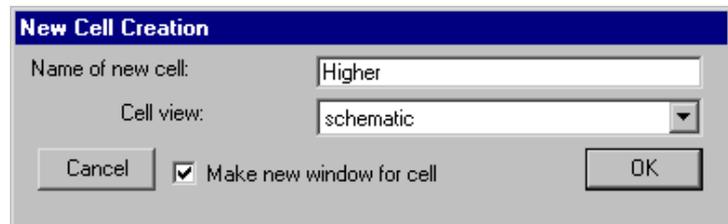
Before creating an icon, there must be connection sites, or *exports* on the schematic. Select the output port of the Buffer node and issue the **Create Export...** command from the **Export** menu. You will be prompted for an export name and its characteristics (the characteristics can be ignored for now).

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.



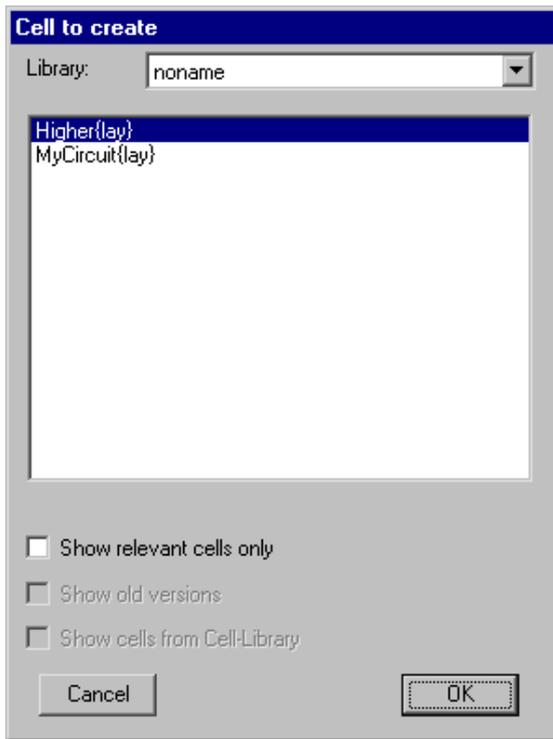
You can now make an icon for this circuit by using the **Make Icon** command of the **View** menu. The icon will be placed in your circuit (you may have to move it away from the rest of the circuitry). The result will look like this.

Now create a new cell with the **Edit Cell...** command of the **Cells** menu. Click on the "New Cell" button at the bottom, and make sure the "Make new window for cell" option is checked in the dialog. Then type the new cell name ("Higher" is used in the example here) and set its view.



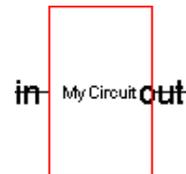
A new (empty) cell will appear in a separate window. Try creating a few simple nodes in this new window (place a gate or two).





Now place an instance of the other cell by using the **New Cell Instance...** command from the **Edit** 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 the **Cells** menu. Note that if the objects in a cell no longer fit in the display window, use the **Fill Window** command from the **Windows** menu.



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. Also, the **Help...** command from the **Info** menu is very useful. It displays a dialog with a list of subjects and offers information about each one.



Chapter 2: BASIC EDITING



2-1: Selection



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.

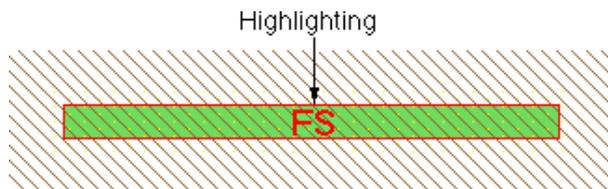
Selection is done with clicks of the *selection* button (this is typically just the left button). Individual nodes and arcs are selected by clicking over them. Once selected, they are highlighted on the screen. If you use the *toggle select* button (typically the same as the *selection* button but with the shift key held), unhighlighted nodes and arcs are added to the selection, but objects that are already highlighted become deselected.

If there are multiple objects under the cursor, use the *select another* button to cycle through them (this is typically the same as the *selection* button but with the control key held). If there are multiple objects under the cursor, and you are trying to do a *toggle select* to add to the selection, then use the *toggle select another* button to cycle through them (this is typically the same as the *selection* button but with the control and shift keys held).

To select an object by its name, use the subcommands of the **Selection** command of the **Edit** menu. The **Select Node...** subcommand selects a node by name, **Select Arc...** selects an arc by name, **Select Network...** selects a network by name, and **Select Export...** selects an export by name. You can also use the Cell Explorer to select from a list of objects (see [Section 3-7](#)).

To select everything in the cell, use the **Select All** subcommand of the **Selection** command of the **Edit** menu. To select everything in the cell that is the same as the currently selected objects, use the **Select All Like This** subcommand of the **Selection** command of the **Edit** menu.

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.

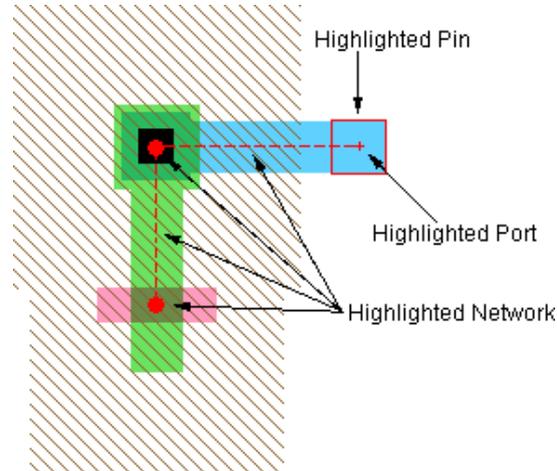
Besides the basic box, there will be other things drawn when an object is highlighted. Highlighted arcs have their constraint characteristics displayed. The example above shows an arc that is both fixed-angle ("F") and



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 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.

Selecting Areas

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 *selection* button where there is no object, and hold it down while dragging over objects, all of those objects will be highlighted. If you need to start the area over an existing object, first deselect everything and then use the *toggle select* button.

To more precisely define a highlighted area, drag a rectangle using the *rectangle select* button. This 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** subcommand of the **Selection** command of the **Edit** menu.

Selecting Text

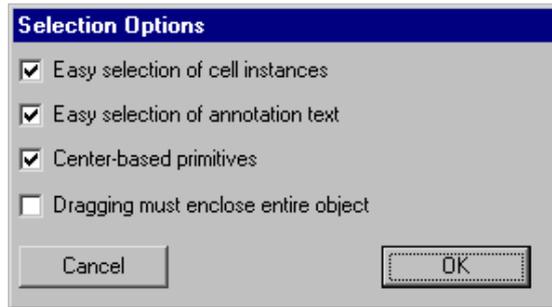
Highlighted text appears as an "X" over the letters. However, text is a special case, so it will not be covered until later ([section 6–8](#)). For now, if you highlight some text, it is best to click again and select something else.

Controlling Selection

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



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. The **Select Nothing** command deselects everything.



To control special selection features, use the **Selection Options...** subcommand of the **Selection** command of the **Edit** menu. The "Center-based primitives" option controls how primitive nodes are created (see [Section 2-2](#)). 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.

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. When objects are not easy-to-select, they require the use of the *special select* button in order to find them (typically the same as the *selection* button but with the Alt/Meta/Option key held). Toggling selection of objects that are not easy-to-select requires the use of the *toggle special select* button. Cycling through multiple objects that are not easy-to-select requires the use of the *special select another* button. Finally, toggling and cycling through objects that are not easy-to-select is done with the *toggle special select another* button.

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)
- Special objects (the text of a cell instance's name)

By default, the first three classes are easy-to-select, and special objects are hard-to-select. If you uncheck "Easy selection of cell instances" in the **Selection Options...** dialog, then cell instances become hard-to-select. If you uncheck "Easy selection of annotation text" in the **Selection Options...** 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 info dialog (use the **Get Info** command of the **Info** menu). If multiple objects are selected, the **Get Info** 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, select them and use either **Make Selected Easy** or **Make Selected Hard**.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 2: BASIC EDITING



2-2: Circuit Creation



Node Creation

Node creation is done by selecting a node from the component menu on the left, or by using one of the **New** commands of the **Edit** menu (**New Cell Instance**, **New Analog Part**, **New Spice Part**, **New Pure Layer Node**, **New Special Object**). These commands then wait for a click in the editing window to place the component. The location of the cursor is aligned to the nearest grid unit, and this adjustment can be controlled with the **Alignment Options...** command of the **Windows** menu.

When placing a node, the cursor points to the *grab point* of the newly created node. By default, this is the center (for primitives) or the lower-left corner (for cell instances). Primitive nodes can use the lower-left corner as the grab-point by unchecking the "Center-based primitives" checkbox in the **Selection Options...** subcommand of the **Selection** command of the **Edit** menu. When this is unchecked, the **Get Info** dialog shows the node's lower-left position, rather than its center.

Cell instances can change their grab-point by placing a Cell-Center node inside of their layout (see [Section 3-3](#)).

Arc Creation

As the introductory example showed, arcs are created by clicking the *creation* 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 *creation* button again runs an arc from the pin to another location. By clicking and holding the *creation* button, you can see the path that the new arc will follow.

If the cursor is over another object when this command is issued, the new wire attaches to that object. To disable this "attach" feature of segment drawing, use the *wire* button instead of the *creation* button.

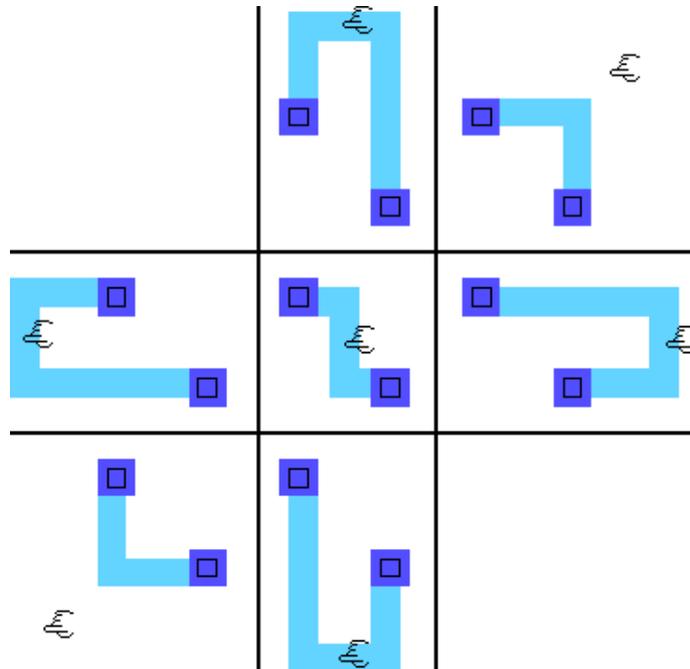
In general, all wiring operations should be done by clicking and holding the *creation* button, then moving the cursor until the intended wiring is shown, and finally releasing. This is recommended because wiring is quite complex and can follow many different paths.

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 *selection* button over the first one, and then using the *toggle select* 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 *toggle*

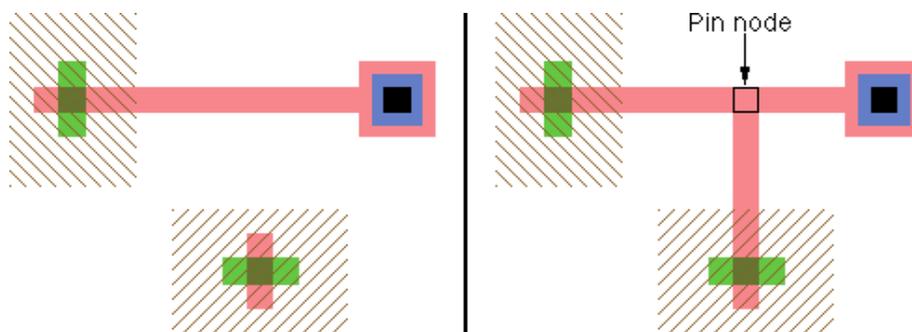


select another button. Once the two nodes are highlighted, use the *creation* button to wire them together. Note that the highlighted ports on the selected nodes are important: arcs will run between them, so they must be compatible in their wiring capabilities.

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 or three arcs and intermediate nodes. The determination of arc location depends upon the position of the cursor when the creation button is clicked. If the cursor is inside of the area defined by the two components, the connection will make a "Z" bend through the cursor. If the cursor is to one side of the components, the connection will make a "U" bend through the cursor location. Finally, if the cursor is in a corner outside of the components, the connection will make an "L" bend towards the cursor.



The nature of the connection can also be affected by the directionality of the ports, which, if it exists, will override cursor location to obtain a sensible connection. Note that for "Z" bends, there are two arcs in one direction, and a third that is perpendicular to them. If there are no other factors (such as directionality), the orientation of the "Z" is made so that the perpendicular arc is shortest.



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.



Whenever an arc is created, the system makes a clicking sound. The sound is a single click for one arc, a double-click for two arcs, and a triple-click for three or more arcs. To disable this sound, use the **General Options...** subcommand of the **User Interface** command of the **Info** menu and uncheck "Click sounds when arcs are created".

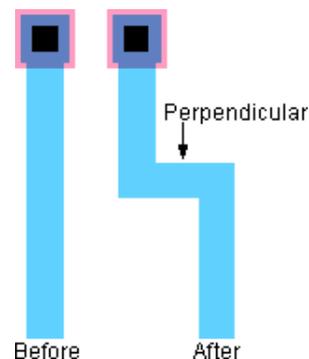
Special Cases

When connecting between two objects that are on different layers, Electric will insert a via to change layers. Only one via can be inserted. Thus, you can wire from a metal-1 pin to a metal-2 pin, and a single via will be created; but you cannot wire from a metal-1 pin to a metal-3 pin, because that would require two vias. When a via is inserted, it is placed closest to the "destination" node (or farthest from the original node).

The width of a new arc is automatically chosen according to a number of factors. The default width is set in the **New Arc Options...** command of the **Arc** menu. 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 **Ends-extend** command of the **Arc** menu.

An unusual circuit creation command is the **Insert Jog In Arc** command of the **Edit** menu. 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** subcommand of the **Cleanup Cell** command of the **Edit** menu, 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.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 2: BASIC EDITING



2-3: Circuit Deletion

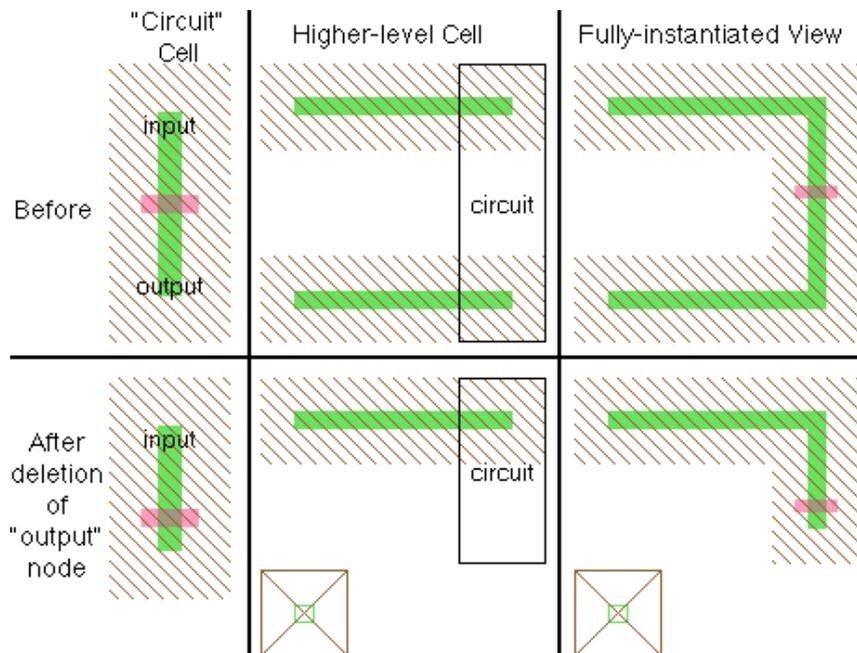


To remove circuitry, select nodes and/or arcs and use the **Erase** command from the **Edit** menu. A keyboard shortcut for this is the Delete key. If there is a highlighted area rather than a highlighted object, everything in the area is erased.

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:

- (1) 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.
- (2) In the interest of cleanliness, if an arc is erased, any isolated pins are also erased.

- (3) If an erased node has an export on it (as in the example below), 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](#)).





Before



After

The **Erase Geometry** 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](#)).

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 2: BASIC EDITING



2-4: Circuit Modification



Movement

Components can be moved by clicking on them with the *selection* 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 menu key is held (the menu key is Command on Macs, Control on Windows and UNIX), then the arrow keys move the object by a block of grid units. A block of grid units is defined in the **Grid Options** dialog to be the distance between bold dots in the grid, initially 10. If you hold both the shift key and the menu 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 subcommands of the **Move** command of the **Edit** menu. The **Quarter Arrow Key Motion** command causes the amount to be quartered (so unshifted arrow keys will move by a quarter lambda). The **Half Arrow Key Motion** command causes the amount to be halved (so unshifted arrow keys will move by a half lambda). The **Full Arrow Key Motion** command causes the amount to be full (so unshifted arrow keys will move by lambda). Note that these menu items are attached to the "q", "h", and "f" keys.

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.

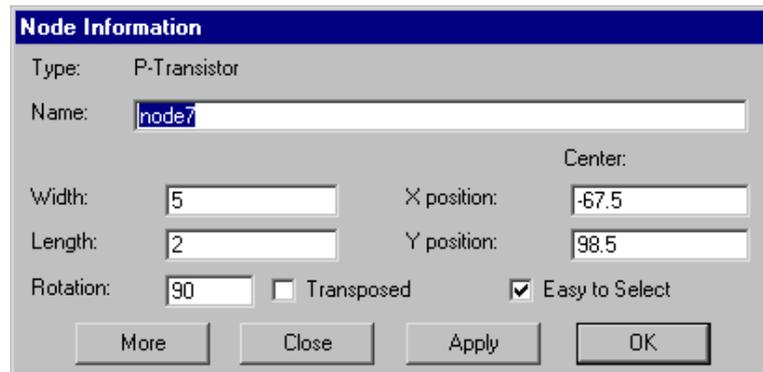
When arcs are moved by a large amount, they cause the connecting nodes to move with them. However, for small arc motion, the arc may shift within its ports. This can only happen if the port has nonzero area and if the arc has the *slidable* constraint (shown with the letter "S" when highlighted). These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



Other Modification

Another way to move a node is to use the **Get Info** command of the **Info** menu, 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 grab-point of the node, which may be in any number of positions (see [Section 2-2](#)). Note that the default unit for typed values is *lambda*, unless another unit is explicitly mentioned (for more information on lambda, [Section 7-2](#)).



The image shows a dialog box titled "Node Information" with a blue header bar. The dialog contains the following fields and controls:

- Type: P-Transistor
- Name: node7
- Width: 5
- Length: 2
- Rotation: 90
- Center: X position: -67.5, Y position: 98.5
- Transposed:
- Easy to Select:
- Buttons: More, Close, Apply, OK

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) can automatically be given names with the **Name All In Cell** and **Name All In Library** subcommands of the **Special Function** command of the **Edit** menu. When naming nodes, the prefix used depends on the node function (see the **New Node Options...** command of the **Edit** menu).

Note the "Easy to Select" checkbox in this dialog. If you uncheck it, you will have to use the [special select](#) button to select the node in the future. This feature allows you to eliminate pieces of circuitry from active editing.

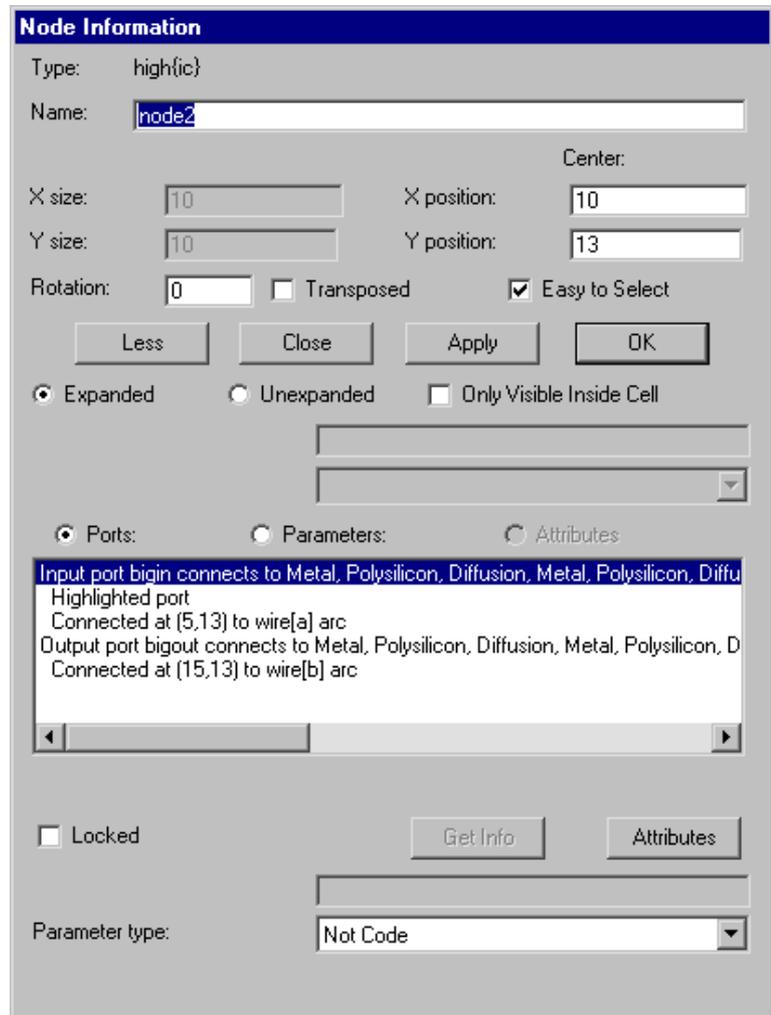
This dialog is modeless: it can remain on the screen while other editing is being done. If a different node is selected, the dialog updates to show that node's information. The "Apply" button changes the selected node to match the new values typed into the dialog.

The **Get Info** dialog can also expand to show more information. When the "More" button is clicked, it grows to full size as shown. To make the dialog default to full size, use the **General Options...** subcommand of the **User Interface** command of the **Info** menu and check "Expandable dialogs default to fullsize".



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

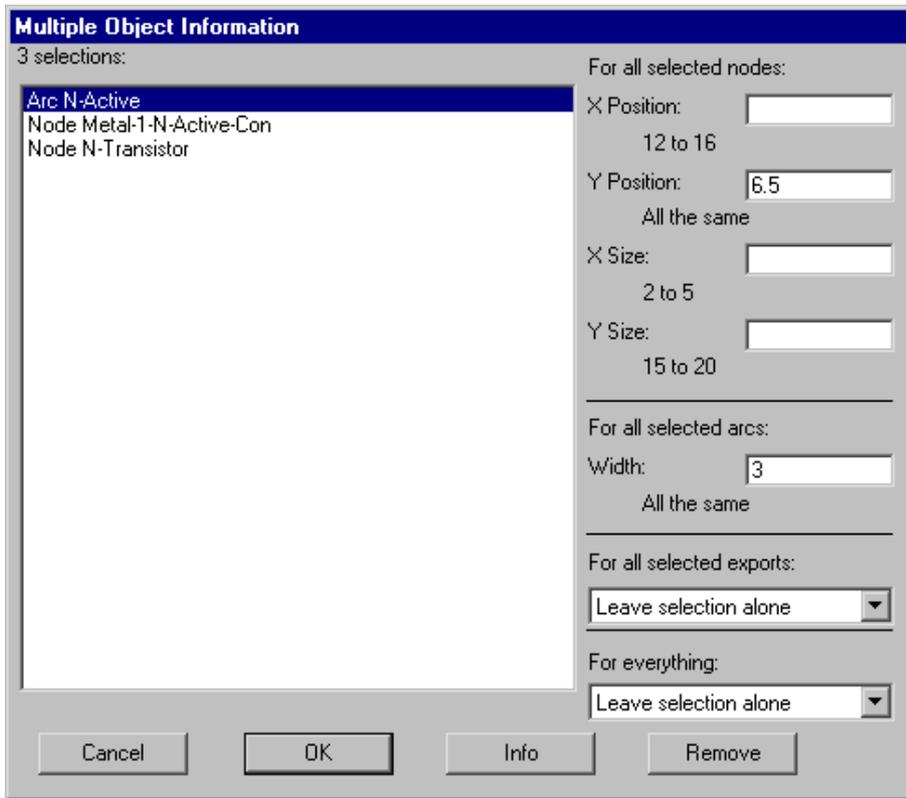
- "Expanded" and "Unexpanded" control how the node is drawn (if it is a cell instance). An expanded instance is one that shows its contents; an unexpanded instance is drawn as a black box.
- "Only Visible Inside Cell" indicates that this node will not be drawn when the current cell is viewed from higher-up the hierarchy.
- "Locked" nodes may not be changed (moved, deleted).
- A scroll area that can view "Ports", "Parameters", 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. If the "Parameters" button is selected, the list shows the parameters on the node. Parameters are similar to attributes: they both are fields on the node. However, parameters are also fields inside of the defining cell. When "Attributes" or "Parameters" are selected, the entries in the list let you modify individual values, and the "Attributes" button brings up an extended dialog for them.



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



If many nodes are selected, the **Get Info** 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 see the full "Get Info" dialog for the selected node, click on the "Info" button. To remove an item from the list (not the circuit, just this list) use the "Remove" button. If only two objects are selected, this dialog shows the distance between their centers.

The multi-object **Get Info** dialog also allows you to change selection style with the "For everything:" popup (see [Section 2-1](#) 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](#) for more on exports).

[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 2: BASIC EDITING

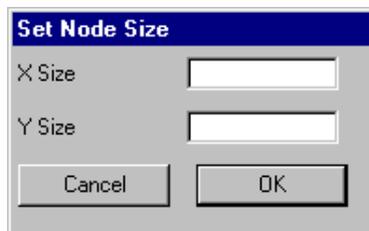


2-5: Changing Size



Node Sizing

To change the size of a node, select it and use the **Interactively** subcommand of the **Size** command of the **Edit** menu. After you do that, your mouse movements will stretch the corner of the node that was closest to it. The opposite corner will remain fixed. Adjust the mouse to the desired position and click. To constrain sizing so that only one dimension changes, hold the Control key while moving the mouse. To abort this operation, type "a".



To change the size of more than one node at a time, select the nodes and use the **All Selected Nodes** subcommand. 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.



Arc Sizing

To change the size of an arc, follow similar procedures. Select the arc and issue the **Interactively** subcommand. 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** subcommand.

Another way to change size is to select the node or arc and use the **Get Info** command of the **Info** menu. Type new X and Y sizes into the dialog for nodes; type a new Width into the dialog for arcs.

The arc **Get Info** dialog also has an "Easy to Select" checkbox, and if you uncheck it, you will have to use the *special select* button to select the arc in the future.

The screenshot shows the 'Arc Information' dialog box with the following details:

- Type: Polysilicon-1
- Network: a2
- Name: [Empty text box]
- Width: 2
- Angle: 180
- Head: Polysilicon-1-Pin
- Tail: P-Transistor
- At: (-210.5,13)
- At: (-193.5,13)
- Bus size: N/A
- Easy to Select:
- Negated:
- Directional:
- Ends extend:
- Ignore head:
- Ignore tail:
- Reverse head and tail:
- Rigid:
- Temporary:
- Fixed-angle:
- Slidable:

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. A CMOS active arc shows highlighting only on its active area, even though the complete arc has implant regions that are much larger. Also note that the X and Y sizes of a node will be swapped if the node is rotated onto its side. Finally, note that the default unit for typed values is *lambda*, unless another unit is explicitly mentioned (for more information on lambda, [Section 7-2](#)).

[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 2: BASIC EDITING



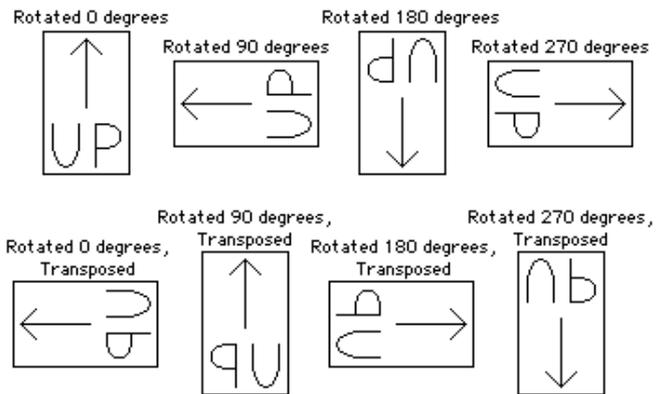
2-6: Changing Orientation



There are two commands that can be used to change the orientation of a node. The **Rotate** command of the **Edit** menu 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 of the **Edit** menu has a submenu that allows you to flip the currently highlighted objects about their vertical or horizontal centerline.

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 rotated and *transposed*. Transposition is a flip about the diagonal centerline. A second mirroring removes the transposition of the node.



Electric stores orientation as a combination of rotation and transposition (this can be seen in the **Get Info** dialog). The diagram on the right illustrates the meaning of rotation and transposition.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 





Chapter 3: HIERARCHY



3-1: Cells

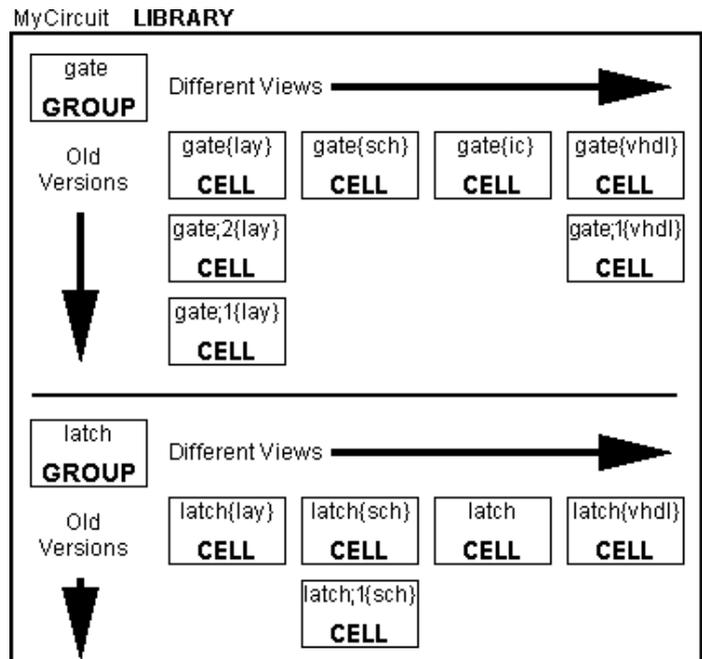


As the introductory examples showed, hierarchy is well supported in Electric. A collection of nodes and arcs is called a *cell*, and instances of cells can be placed in other cells. Thus, Electric still manipulates nodes and arcs, but the nodes come in two forms: *primitive* and *complex*. Primitive nodes are found in the component menu and are pre-defined by the technologies. Complex nodes are actually instances of other cells, and are found in libraries.

Besides organizing cells into a hierarchy, Electric also organizes cells according to their *view* and *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 cell's view. 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 cells: "gate" and "latch". In this library are cells called "latch", "latch;1{sch}", and "latch{lay}", all of which are cells of the cell called "latch". The cell "latch" has no view name and is therefore from the "unknown" view of the cell. The cell "latch{lay}" is the layout view of the cell.

To rename a cell, use the **Edit Cell...** command of the **Cells** menu, which has a rename area at the bottom of the dialog.

[Previous](#)

[Table of Contents](#)

[Next](#)



Chapter 3: HIERARCHY

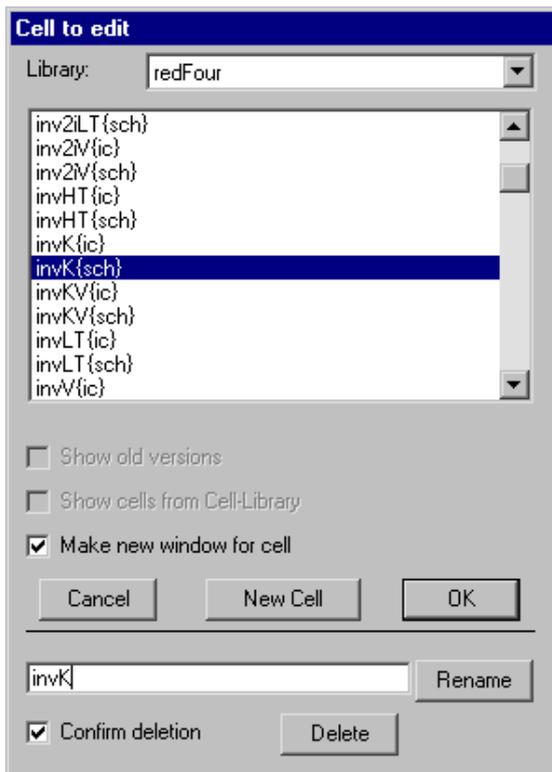


3-2: Creating and Deleting Cells



Cell Creation and Deletion

Cells are created with the **Edit Cell...** command of the **Cells** menu.

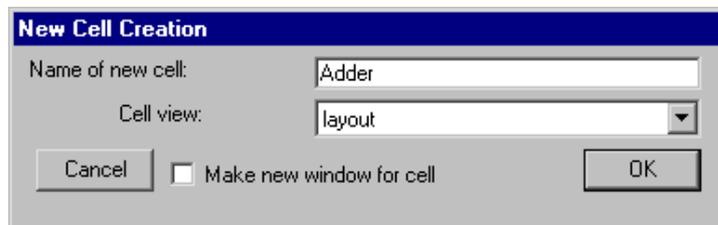


When the **Edit Cell...** command is used, a dialog of existing cells is presented. You can choose to show the cell in the current window, or create a new one.

The bottom part of the dialog lets you rename and delete cells.

In addition, there are two options for filtering the list of cells. If there are cells with multiple versions, you can choose to see the old versions or ignore them. If there are cells from a "cell library", you can remove them from the list so that it shows only your design hierarchy.

To create a new cell, click on the "New Cell" button. This then presents a dialog in which the new cell name and its view can be specified.



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.

There are two ways to make a copy of a cell. The **Duplicate Current Cell** command of the **Cells** menu copies the cell in the current window to a new cell with a new cell name. You will be prompted for the new name. The **New Version of Current Cell** command also makes a copy of the cell in the current window. However, this copy is a "new version", which has the same cell name. The newly created cell is displayed in the window. Both of these commands work within the same library.

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...** subcommand of the **Special Cell Lists** command of the **Cells** menu.

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 of the **Cells** menu. 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.

 [Previous](#)  [Table of Contents](#) [Next](#) 



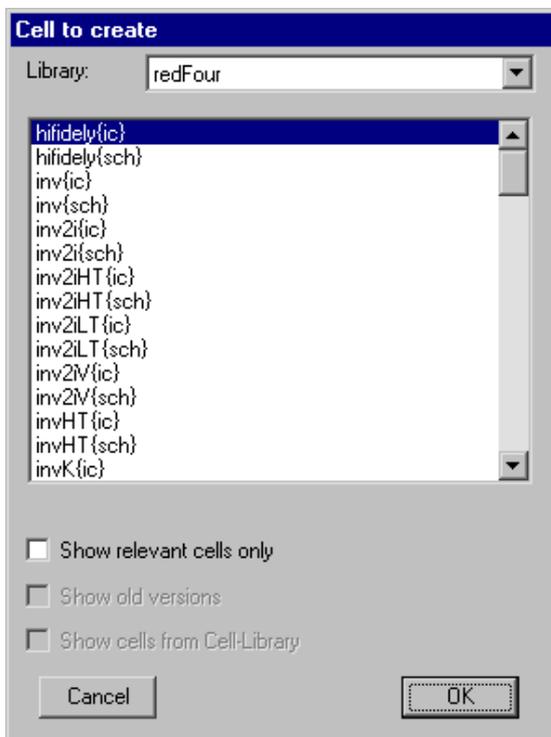
Chapter 3: HIERARCHY



3-3: Creating Instances



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



Another way to place an instance of a cell is to use the **New Cell Instance...** command of the **Edit** menu. 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.

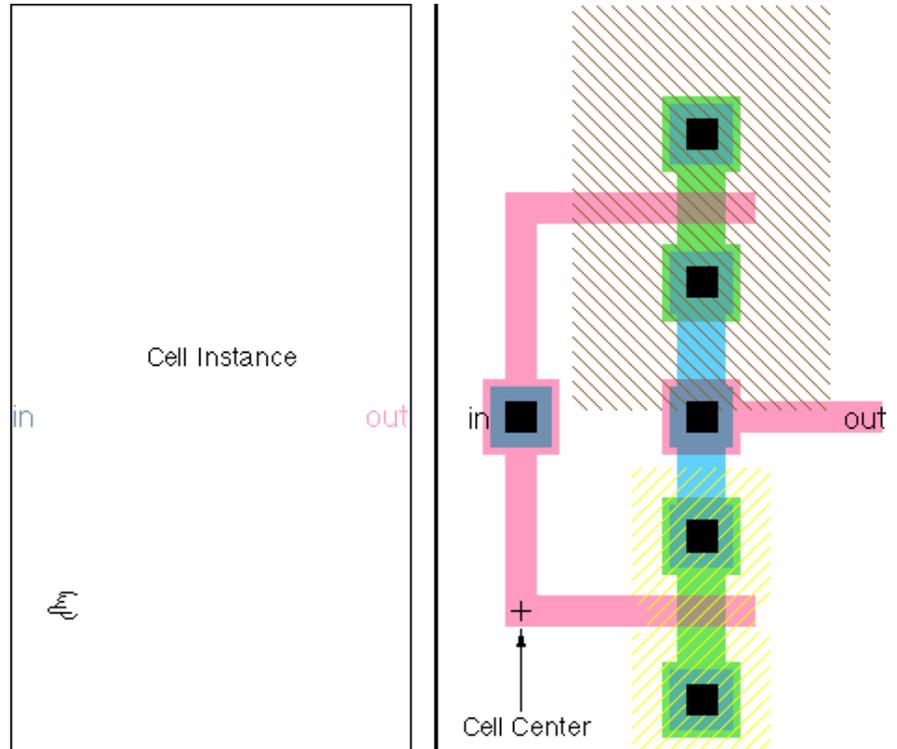
Three checkboxes allow you to limit the list of cells. By checking "Show relevant cells only" only those cells that have the same view type as the current one will be shown. By unchecking "Show old versions" the list will remove old versions of cells. By unchecking "Show cells from Cell-Library" the list will remove cells that come from a cell library.

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](#) 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. Highlight the existing cell and use the **Duplicate** command of the **Edit** menu. 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 *grab point*. Normally, the grab point is at the lower-left corner, but it can be modified by the placement of a *Facet-Center* node inside of the cell. To place this, use the **Cell Center** subcommand of the **New Special Object** command of the **Edit** menu. You must place it inside of the cell definition, and it affects the grab point for all subsequent creation of cell instances.



[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 3: HIERARCHY



3-4: Examining 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 subcommands of the **Expand Cell Instances** command of the **Cells** menu. The **One Level Down** subcommand opens up the next closed level; the **All the Way** subcommand 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 subcommands of the **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** subcommand closes up the bottommost expanded level; the **All the Way** subcommand closes all levels from the bottom; and the **Specified Amount...** lets you type a number of levels of hierarchy to close.

The expansion information can also be set or reset by using the **Get Info** command of the **Info** menu 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 of the **Cells** menu 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** subcommands and the **Look Inside Highlighted** command. The **Expand Cell Instances** subcommands 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 the *rectangle select* button, to show only a specific part of the circuit.

Examining cell instances can be quite time consuming, especially if you are looking at an entire chip, fully flattened. To avoid delays, Electric simplifies the display of cells when the scale grows too tiny. These tiny cells get "hashed-out" rather than fully drawn. See [Section 3-7](#) for more information.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 3: HIERARCHY



3-5: Moving Up and Down the Hierarchy

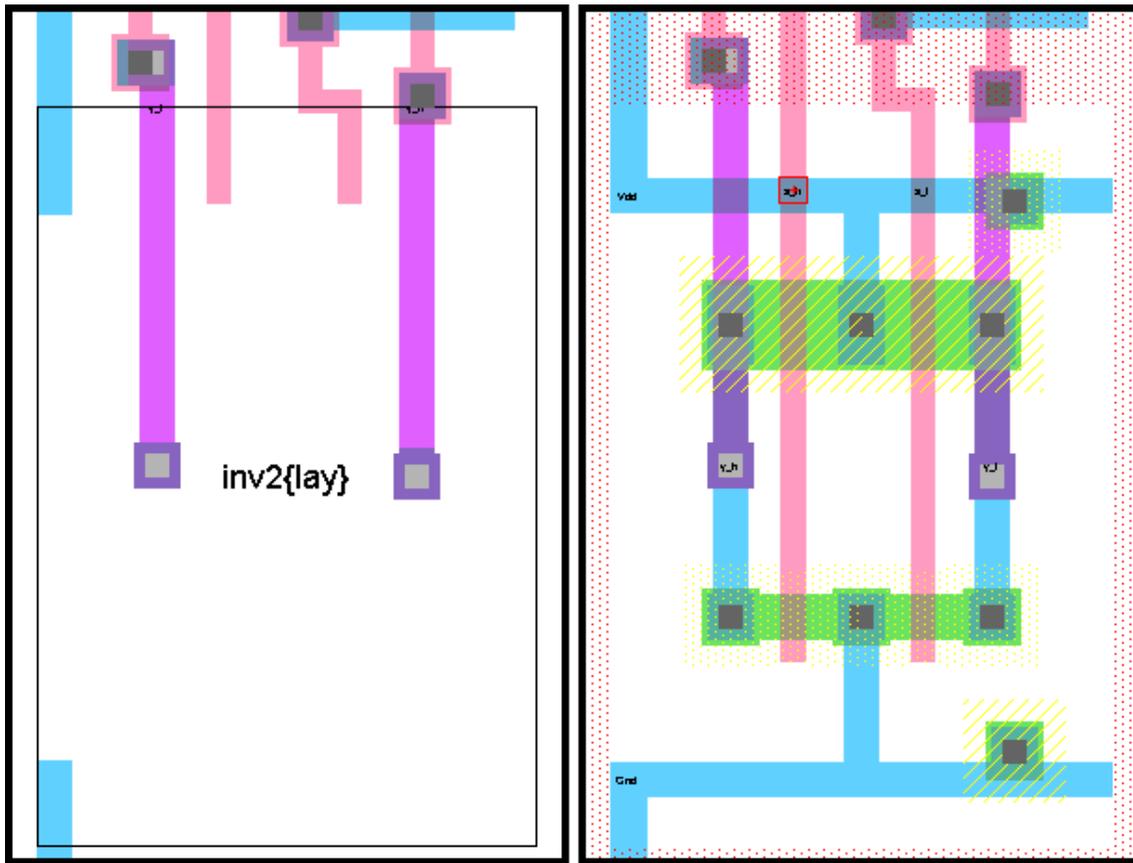


Each editing window in Electric displays a single cell. Editing changes can be made only to that cell, and not to any subcells that appear as instances. Thus, you may be able to see the contents of a cell instance, but you cannot edit it.

To edit a cell instance, use the **Down Hierarchy** command of the **Cells** menu. 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 the instance is rotated or mirrored, going **Down Hierarchy** will show it in its original, untransformed orientation. The command **Down Hierarchy in Place** does not have this limitation: if an instance is rotated or mirrored, this command allows you to edit the cell in that orientation. Also, the **Down Hierarchy in Place** command shows the upper level of hierarchy that surrounds the instance (although it is shaded because you cannot edit it). In the figure below, the left side is the upper-level of hierarchy, with an instance shown. The right side is the result of **Down Hierarchy in Place**, showing the upper level shaded.





If an icon is selected, the **Down Hierarchy** 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.

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** command was 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.

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.

[← Previous](#)
[↑ Table of Contents](#)
[Next →](#)



Chapter 3: HIERARCHY



3-6: Exports



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 of the **Export** menu. To see all exports that have been defined in the current cell, use the **Show Exports** command of the **Export** menu. 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 create an export, select a port on a node and use the **Create Export...** command of the **Export** menu. The resulting dialog requests an export name and some characteristics.

The dialog box titled "Create Export on Highlighted Node" contains the following fields and controls:

- Export name:** A text input field containing "a1".
- Export characteristics:** A dropdown menu currently set to "Unknown".
- Always drawn:** An unchecked checkbox.
- Body only:** An unchecked checkbox.
- Reference export:** An empty text input field.
- Buttons:** "Cancel" and "OK" buttons at the bottom.

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 indexed 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 three 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 of the **Export** menu, 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 Highlighted**. 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 9-5](#) for more on automatic wiring.

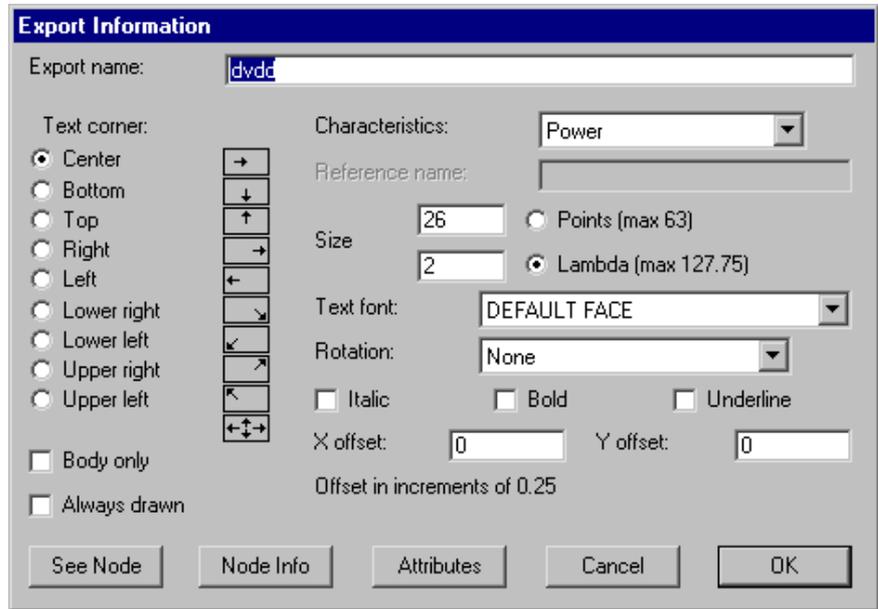
Another special case in export creation is the **Add Export from Library...** command, which copies exports from another library to the current one. The other library is examined for cells whose names match ones in the current library. When a cell is found in the other library, all of its exports are 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](#) on Standard Cell Libraries). Because some formats contain geometry and others contain connectivity, this command is needed to put them together.



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 Cell Explorer's contents view (see [Section 3–7](#)) or by using the **Select Export...** subcommand of the **Selection** command of the **Edit** menu.

Once a port has been exported, its characteristics can be modified by selecting the export name and using the **Get Info** command of the **Info** menu. You can change basic export information such as the name, characteristic, and reference name (if applicable). You can also change the appearance of the export by editing the size, font, style, placement, and rotation of the name. See [Section 6–8](#) for more about text appearance.



Buttons at the bottom of the Export Get Info dialog allow you to examine related objects. The "See Node" button shows the node on which this export resides, and the "Node Info" button brings up a **Get Info** dialog for that node. The "Attributes" button brings up an Attributes dialog for the export (see [Section 6–8](#) for more on Attributes).

You can change the characteristics of many exports by selecting them and setting the "For all selected exports:" popup in the **Get Info** dialog of the **Info** menu. You can change the name of exports by using the **Rename Export...** command of the **Export** menu.

Ports and exports can be displayed on the screen in many different ways. The **Port and Export Options...** command of the **Export** menu provides three options: "Full Export Names" shows full text names, "Short Export Names" shows export names only up to the first nonalphabetic character, and "Exports as Crosses" shows crosses at the locations.



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 **Layer Visibility** command of the **Windows** menu.

Export Deletion and Movement

You can delete an export simply by selecting its name and typing the delete key. You can also use the **Delete Export** command of the **Export** menu.

To remove many exports at once, the **Delete All Export on Highlighted** command removes all exports on all highlighted nodes. Also, the **Delete All Export in Area** command removes only those exports that are in the selected area (use the *rectangle select* button to define a precise 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 **Port and Export Options...** dialog, 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 of the **Export** menu can be used. Before using this command, two nodes and their ports must be highlighted with *selection* button and *toggle select* button. The export is moved from the first node to the second node.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 3: HIERARCHY



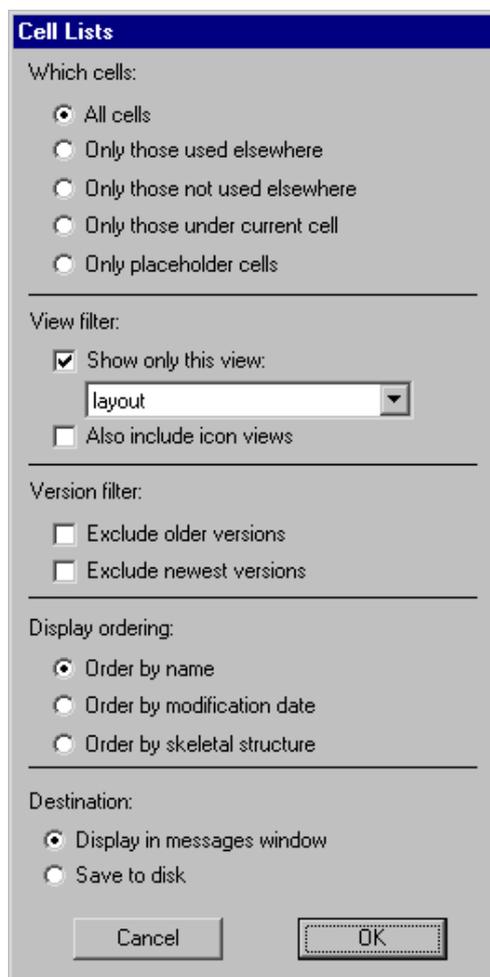
3-7: Cell Information



Miscellaneous Commands

The most basic cell information that you can get is a list of cells. This can be done with the **Edit Cell...** command of the **Cells** menu. The dialog will show the cell names (at which point you can cancel the dialog).

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



To get such information for a subset of cells, 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).

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



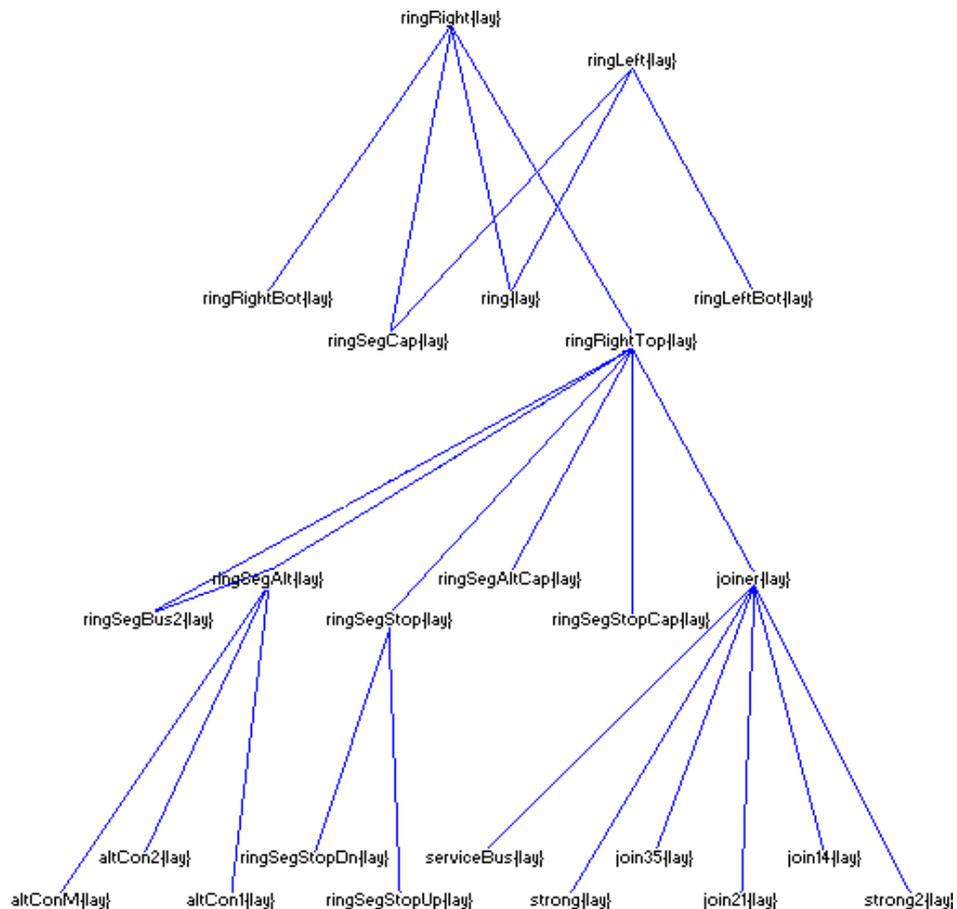
(tab-separated).

The last six columns show the usage and five 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 of the cell are locked), "C" (if the cell is part of a cell library), "D" (if the cell has passed design-rule checking), and "N" (if the cell has passed network consistency checking).

To see all exports in the current cell, use the **Export** menu commands **List Exports** (for a text listing) or **Show Exports** (for a graphical indication). 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 from the **Info** menu.

For more cell information, use the subcommands of the **Special Cell Lists** submenu of the **Cells** menu. The **List Nodes in this Cell** subcommand shows all nonprimitive nodes in the current cell. The **List Cell Instances** subcommand shows all cell instances below the current cell. The **List Cell Usage...** subcommand looks up the hierarchy and finds cells that contain the a particular cell as an instance (you will be prompted for the particular cell).

The **Graphically, Entire Library** subcommand displays a graph of every cell in the library. The **Graphically, From Current Cell** subcommand 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 with the other cells. If you select a name in this graph, then the **Edit Cell...** command of the **Cells** menu defaults to that cell.



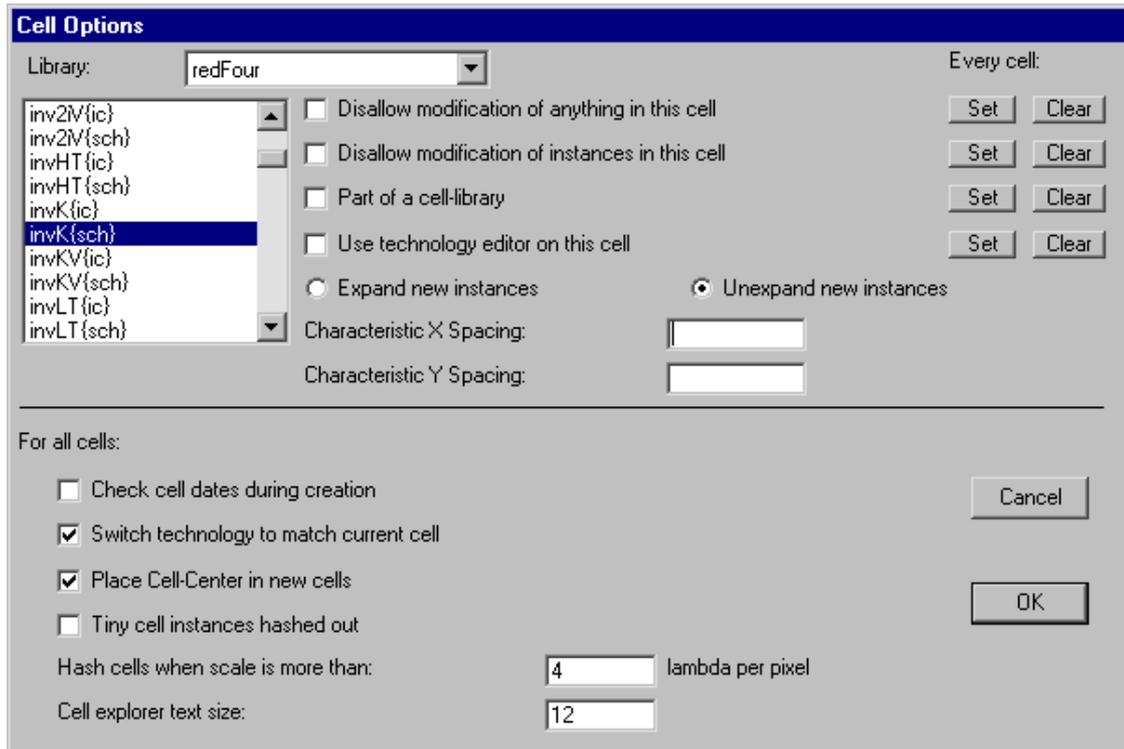
The **Edit Documentation View** command of the **View** menu allows you to store arbitrary text with the cell. The text editing window that appears can contain any information. See [Section 4-10](#) for more on text editing.

The **List Layer Coverage** command of the **Info** menu computes the percentage of the cell that is covered by each layer. This is useful information for certain fabrication processes which insist on a minimum amount of each layer in the die.



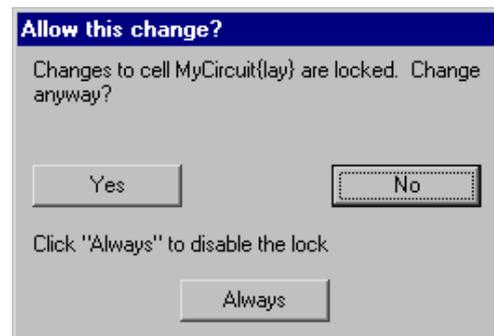
Cell Options

To examine and set more information about existing cells, use the **Cell Options...** command of the **Cells** menu:



The upper part of the dialog provides options on a per-cell basis (choose the library and the cell and then set its options). 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 sub-instances are locked. This is useful when you have a correct instance placement and are doing wiring. Buttons on the right allow you to set or clear these flags for all cells.

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, disallow the change, or remove the lock (which effectively unchecks the locks in the **Cell Options...** dialog).



The check box "Part of a cell-library" indicates that this cell is from a library of standard cells and should be treated accordingly. Buttons on the right allow you to set or clear this flag for all cells. Most dialogs that list cells can remove those from cell-libraries to keep the list simpler.



The check box "Use technology editor on this cell" indicates that this cell part of a technology editor library. Editing of cells in a technology editor library is done in a special mode (indicated with a yellow border in the window). See [Section 8–2](#) for more information about technology editing.

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](#)).

The lower part of the dialog contains options that will apply to all cells. The check box "Check Cell dates during creation" 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 the **Cells** menu.

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 (named "Facet–Center" for historical reasons) placed at the origin (see [Section 3–3](#) for more on Cell centers).

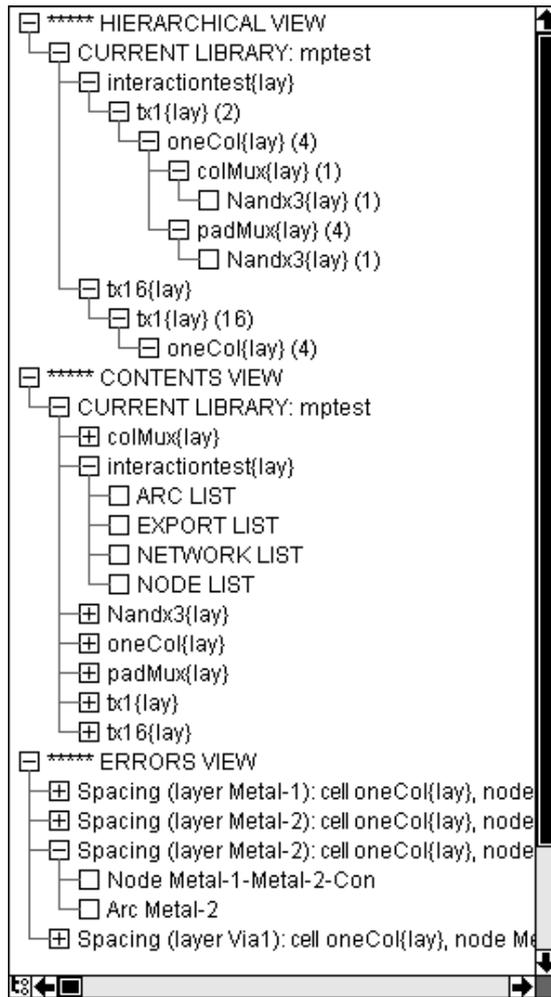
The check box "Tiny cells hashed out" requests that cells be displayed with a gray hash pattern when they are too tiny for their contents to be distinguished. Without this option, all geometry is drawn, which can take a long time. You can control the threshold of "tinyness" by setting the number of lambda units per pixel. Any view in which this many lambda units are shown in pixel will be too small to display fully.

The Cell Explorer

The **Cell Explorer...** command of the **Cells** menu splits the current window, and shows a hierarchical "explorer" window in the left half. Another way to get the cell explorer is to click on the "tree" icon in the lower–left of the window.

The explorer window has 3 sections: the Hierarchical View, Contents View, and Errors View.





The Hierarchical View of the cell explorer lists only the "top level" cells of each library (top level cells are those that are not used as instances in any other cells). By clicking on the box with the "+", the cell's contents are shown in a recursively indented manner. Each cell lists the number of times that it occurs in the higher-level cell.

The Contents View of the cell explorer lists every cell in every library. Inside of each cell is a list of arcs, exports, networks, and nodes. When editing technology-edit libraries, the Contents View breaks down the library according to layers, nodes, and arcs.

The third part of the Cell Explorer is the Errors View. This lists all errors that were generated by other tools (DRC, ERC, NCC, etc.) and which are normally presented with the "" keys.

Any text shown in the Cell Explorer can be copied (and pasted to another document) by selecting a line and using the **Copy** command. In both the Contents and Hierarchical View, you can double-click on any object to see it in the other half of the window. Double-clicking on a cell shows that cell. Clicking a cell and typing the DELETE key deletes that cell. Double-clicking on a library name makes that library the current one.

In addition, the Cell Explorer uses a "context menu" that is accessed by right-clicking on an entry (use command-click on the Macintosh). The context menu offers special operations, for example the ability to recursively open or close all entries below the current one.

The line which separates the Cell Explorer from the edit window can be moved simply by clicking on it and dragging it (see [Section 4-3](#) for more on window control). To make the Cell Explorer go away, delete that window partition, click on the tree icon again, or issue the **Cell Explorer...** command again.

The font size of text in the Cell Explorer can be set in the "Cell explorer text size" field of the **Cell Options...** dialog.



Chapter 3: HIERARCHY



3–8: Rearranging Hierarchy



Creating New Levels of Hierarchy

In order to manipulate hierarchical circuits, it is useful to create and delete levels of the hierarchy. The **Package into Cell...** command of the **Cells** menu 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 *rectangle select* button and drag a rectangle. 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.

Removing Levels of Hierarchy

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.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 3: HIERARCHY



3–9: Libraries



A *library* is a collection of cells and 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 of the **File** menu. To see which libraries are read in, use the **List Libraries** command.

To create a new, empty library, use the **New Library...** command of the **File** menu. 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 **New Cell Instance...** command of the **Edit** menu). 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 of the **Cells** menu and select "Only placeholder cells".

Reading Libraries

The **Open Library...** command of the **File** menu brings a new library into Electric from disk. These disk files are in a private binary format (that is, not readable outside of Electric).

Besides Electric libraries, it is possible to read circuit descriptions that are in other formats with the **Import** command of the **File** menu. Most of these commands place the data into a new library that has the same name as the disk file. When reading these files, it is important that the current technology be set to the one in the file so that proper layer conversion can be done (use the **Change Current Technology...** command of the **Technology** menu). These formats can be read:

- **Caltech Intermediate Format (CIF)** 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. Use the **CIF Options...** subcommand of the **IO Options** command to affect how CIF is read.
- **Stream (GDS II)** 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. Use the **GDS**



Options... subcommand of the **IO Options** command to affect how GDS is read. On Windows, it is possible to select multiple files for input. If you do this, all of the GDS will be read into the current library.

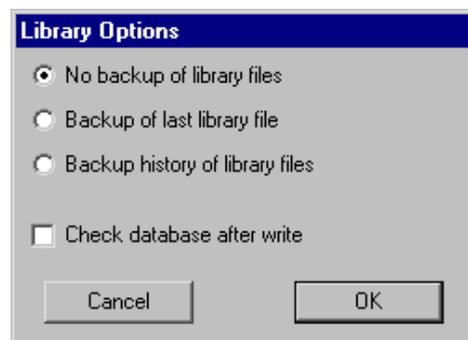
- **Electronic Design Interchange Format (EDIF)** is used to describe both schematics and layout. Electric reads EDIF version 2 0 0.
- **VHDL** is a hardware description language that describes the structure of a circuit. When VHDL is read, it is not converted to layout, but instead is placed into a text cell with the {vhdl} view.
- **Library Exchange Format (LEF)** is an interchange format that describes the cells in a library. The cells that are read in contain ports, but very little contents.
- **Design Exchange Format (DEF)** 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 Options...** subcommand of the **IO Options** command to affect how DEF is read.
- **AutoCAD DXF** 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 (you must switch to the Artwork technology before importing DXF). Use the **DXF Options...** subcommand of the **IO Options** command to affect how DXF is read.
- **Standard Delay Format (SDF)** is used to read test vector parameters and place them on cells in the current library. Before this data can be used by the simulator, one of the three sets (**Typical**, **Minimum**, or **Maximum**) must be selected with the **Annotate Delay Data (ALS)** subcommand of the **Simulate** command of the **Tools** menu.
- **Schematic User Environment (SUE)** 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).
- **Readable Dump** is an Electric-specific format that captures the entire database, but in an editable text format. Because it is text, it is slower to read than Electric's binary files, and takes up more space on disk. However, it can be transferred between machines more reliably and can be edited if necessary.

See [Section 7-3](#) for more information on external formats.

Writing Libraries

Writing libraries to disk is done with the **Save Library** command of the **File** menu. The **Save All Libraries** command of the **File** menu writes all libraries that have changed. 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.

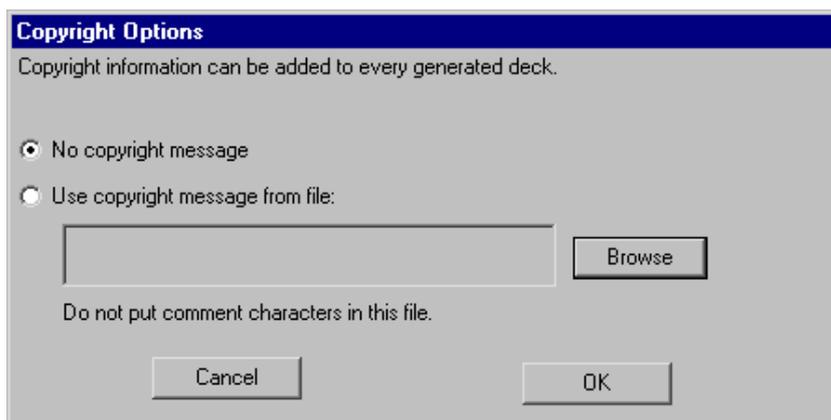
The **Library Options...** subcommand of the **IO Options** command of the **File** menu controls the writing of libraries to disk. By default, saved libraries overwrite the previous libraries and no backup is created. If you choose "Backup of last library file", then the former library is renamed so 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. You can also use this dialog to request that the database be checked when saves are done.



Electric can also write external format files with the **Export** command of the **File** menu. These formats can be written:

- **Caltech Intermediate Format (CIF)** 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 Options...** subcommand of the **IO Options** command to affect how CIF is written.
- **Stream (GDS II)** 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 Options...** subcommand of the **IO Options** command to affect how GDS is written.
- **Electronic Design Interchange Format (EDIF)** can write either the Netlist or the Schematic view of the circuit. Use the **EDIF Options...** subcommand of the **IO Options** command to affect how EDIF is written. Electric writes EDIF version 2 0 0.
- **Library Exchange Format (LEF)** is an interchange format that describes the exports on cells in a library.
- **Circuit Design Language (CDL)** is a Cadence interchange format for netlists.
- **Eagle** is an interface to the Eagle schematics design system (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. Use the **Define...** subcommand of the **Attributes** command of the **Info** menu to create these attributes (see [Section 6–8](#) for more information). Also, every network must be named.
- **ECAD** is an interface to the ECAD schematics design system (netlist format).
- **Pads** is an interface to the Pads schematics design system (netlist format).
- **AutoCAD DXF** is a solid–modeling interchange format. Use the **DXF Options...** subcommand of the **IO Options** command to affect how DXF is written.
- **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.
- **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 **Print Options...** command of the **File** menu.
- **HPGL** is the Hewlett–Packard printing language. The output file contains only a visual representation of the current cell (or part of that cell). HPGL options can be controlled with the **Print Options...** command of the **File** menu.
- **Readable Dump** is an Electric–specific format that captures the entire database, but in an editable text format. Because it is text, it takes up more space on disk. However, it can be transferred between machines more reliably and can be edited if necessary.

The exported files from Electric are often considered to be proprietary information, and must be marked appropriately.



By using the **Copyright Options...** subcommand of the **IO Options** command of the **File** menu, a disk file with copyright information can be inserted into exported files. Since each export file has a different format for comments, the copyright file should not contain any such characters. Instead, the system will insert the proper comment characters for the particular export



format.

The copyright file will be inserted into decks exported for CIF, LEF, and CDL, as well as in simulation netlists for Verilog, SPICE, FastHenry, IRSIM, and ESIM. See [Section 7–3](#) for more information on external formats.

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 libraries, 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 (with the silicon compiler) 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)** subcommand of the **Import** command of the **File** menu. 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)** subcommand of the **Import** command of the **File** menu. Note that the proper GDS layers must be established first (with the **GDS II Options...** subcommand of the **IO Options** command of the **File** menu). As an aid in this process, you may find it helpful to read either the file "tsmc25.txt" or "umc18.txt" in the Electric library directory (these are Readable Dump files that can be read with the **Readable Dump** subcommand of the **Import** command of the **File** menu). Note that 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 of the **File** menu if it is not). To merge the LEF port information, use the **Add Export from Library...** command of the **Export** menu. You will be prompted for a library, and should select the one with the LEF data.
 - At this point, the GDS library now has standard cells in it. Before saving it to disk, you should probably use the **Cell Options...** command of the **Cells** menu and set all of the cells to be "Part of a cell–library".
-

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 3: HIERARCHY



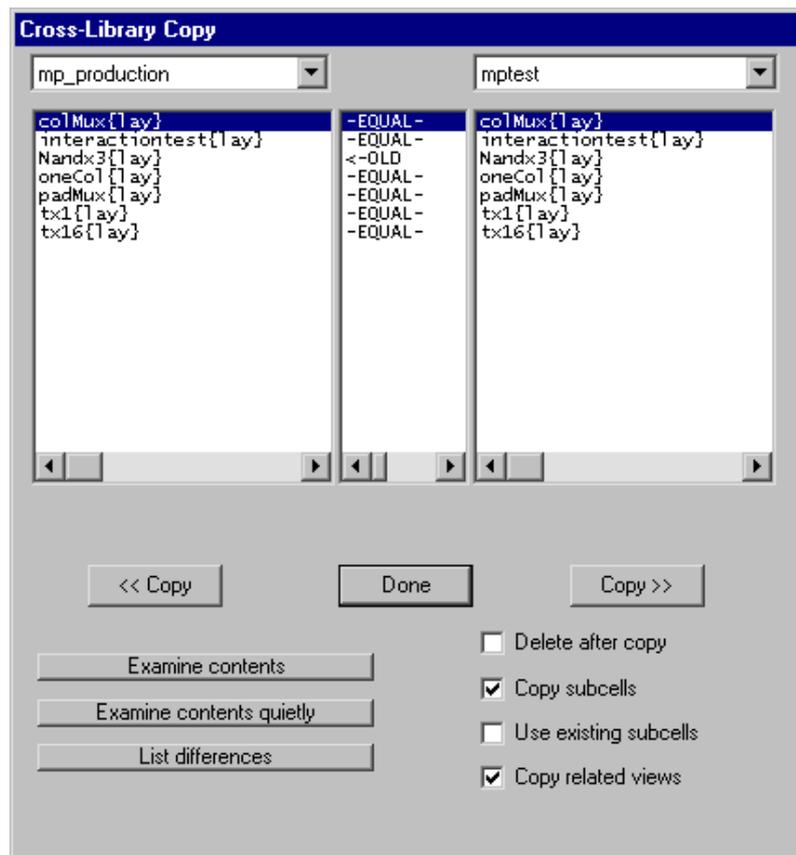
3-10: Copying Between Libraries



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 of the **Cells** menu 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 modification dates to determine which is newer. The "Examine Contents" button compares the contents of cells whose dates are different and displays an indication of whether they are actually different or just out of date. The "Examine Contents Quietly" button does the same thing, but it does not explain which differences have been found. The "List differences" button shows all differences in the messages window.



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 transferred. By checking "Copy related views", all related views (icon, schematic, layout, etc.) are also transferred.

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".
-

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 3: HIERARCHY



3-11: Cell Views

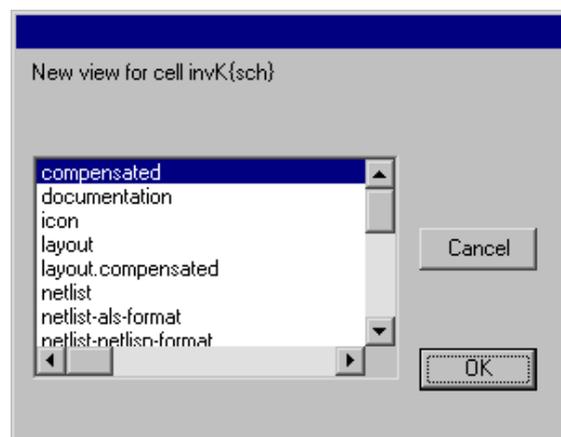


Setting a Cell's View

Each cell has a *view*, which provide 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)
- "schematic-page-N" (for multipage schematics)
- "icon" (to describe a cell symbolically)
- "simulation-output" (captured waveforms)
- "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 **Edit Cell...** command, you can specify its view in the "New Cell" subdialog. After creation, you can change the current cell's view with the **Change Cell's View...** command of the **View** menu. Note that this is one of the few commands in Electric that is NOT undoable.



Switching between Views of a Cell

When editing one view of a cell, there are a set of 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 unnumbered schematic view.
- Use **Edit Multi–Page Schematic View...** to switch to a particular page of a multipage schematic (this is the only command that can create new pages of a multipage schematic).
- 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.

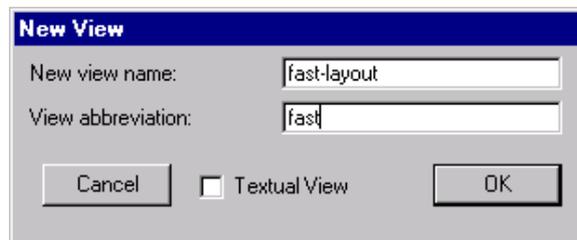
For all other view types, use **Edit Other View...** and select the desired view. Note that these commands are equivalent to the **Edit Cell...** command of the **Cells** menu with an appropriate selection.

When editing cells with text–only views, the window becomes a text editor. You may then use the **Write Text Cell...** and **Read Text Cell...** commands of the **Cells** menu 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. Thus, **Make Multi–Page Schematic View...** creates a new page in a multi–page schematic. **Make Documentation View** creates a blank documentation cell. **Make Other View...** prompts for any view type and creates a cell with that view.

Creating and Deleting Views

If the list of possible views is not sufficient to describe a cell, new views can be created with the **New View Type...** command of the **View** menu. This command requests a name and an abbreviation.



Generally, an 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".

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

To delete a view, use the **Delete View Type...** command of the **View** menu. 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.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 3: HIERARCHY



3-12: Automatic View Generation



Electric has facilities for automatically converting from one view to another. When converting the current cell, the newly generated cell is shown in a new window.

Conversion between Layout and Schematic

To convert from IC layout to a schematic, use the **Make Schematic View** command of the **View** menu. This builds a new cell using analog schematic parts that are equivalent to the IC components in the current cell. If there is already a schematic view of the current cell, a new version of that view is created.

To convert between different IC layout technologies or from schematics to layout, use the **Make Layout View...** command of the **View** menu. You will be prompted for the technology to use for the new cell. If both the current and the new cell are of the Layout view, the new cell becomes a newer version. Conversion between layout technologies is useful when the fabrication process or design rules have changed, but it can only be done within a similar family of technologies, i.e. between CMOS technologies. When converting from schematics to layout, all wires appear as Universal Arcs from the Generic technology. These must be converted to layout manually.

Skeletonization

The **Make Skeleton View** command of the **View** menu converts the current cell into a new, skeleton view. Skeleton cells contain only the important parts (the nodes with exports and a few others that define the boundary). When skeletonizing, the new cell has information stored on it that points to its original cell, so that it can be restored to its complete geometry later. (At this time, no automatic facilities exist for restoring skeleton cells. Use the **Change...** command of the **Edit** menu and select the nonskeleton cell.) Skeletonizing is useful when libraries get to be very large, for they allow entire levels of hierarchy to be abstracted, with their actual contents kept in another library.

Icons

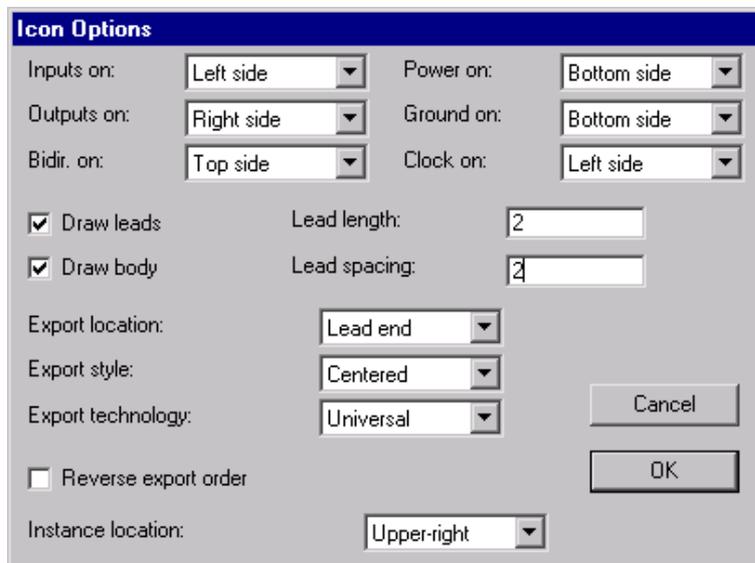
A particularly useful view type is icon. The icon cell is used for instances of an associated *contents* cell, which contains schematics. For example, you may have a cell called "adder{sch}" which contains a schematic. You may then create a cell called "adder{ic}" that contains a circle with a plus sign inside (these are nodes in the Artwork technology). This is then the icon for the layout 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 of the **View** menu. 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 Options...** command of the **View** menu. This command lets you place each of the different export types on any of the four sides of the icon.

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). 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).



You can choose whether or not to draw the leads and the body of the icon. You can set the size and spacing of the leads. 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).

The icon cell is correctly tied to its contents in most respects. If you descend into it (with the **Down Hierarchy** command of the **Cells** menu), 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").

VHDL

The **Make VHDL View** command of the **View** menu converts the current cell into a VHDL textual description. All subcells used in the current one are also converted. By default, the VHDL is placed in cells with the "vhdl" view. However, by unchecking the "VHDL stored in cell" item of the **VHDL Options...** dialog (in the **VHDL Compiler** subcommand of the **Tools** menu), the VHDL will be placed in individual disk files. Recheck the entry to place VHDL in cells again. See [Section 9–10](#) for more on Electric's VHDL facility.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 4: THE DISPLAY



4-1: Introduction to the Display



Nearly every aspect of the display can be controlled. There are commands to manipulate the editing windows, menus, colors, and more.

One useful feature of the display is that drawing can be aborted by typing the interrupt key (Ctrl-C on UNIX,  C on Windows,  on the Macintosh; see [Section 1-9](#)). This helps if a long and complex display operation begins that need not finish.

Electric is able to display multiple editing windows, and these windows can be subdivided repeatedly.

If a window appears incorrect, you can redraw it with the **Redisplay Window** command of the **Windows** menu.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 4: THE DISPLAY



4-2: The Messages Window



Before describing the commands for controlling the editing windows, it is useful to mention the messages window, which contains scrollable text.

The messages window is the place for all textual information. If the window is closed, it will go away, but it will reappear whenever messages are displayed. If this window is obscured by other Electric windows, it will pop to the top when new information is sent to it.

The text in the messages window can be selected with the cursor and edited with the **Cut**, **Copy**, and **Paste** commands of the **Edit** menu. You can remove all text with the **Clear** subcommand of the **Messages Window** command of the **Windows** menu. You can change the font and text size of the messages window with the **Set Font...** subcommand of the **Messages Window** command of the **Windows** menu (which has a different format on each platform).

The text in the messages window can be saved to disk by using the **Save Messages** subcommand of the **Messages Window** command of the **Windows** menu (the text is saved in the file "emessages.txt").

The location of the messages window can be saved with your options by using the **Save Window Location** subcommand of the **Messages Window** command of the **Windows** menu. This command is not needed on the Macintosh, because all changes to Macintosh windows are remembered without request.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 4: THE DISPLAY



4-3: Creating and Deleting Editing Windows



Multiple 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 **Edit Cell...** command of the **Cells** menu. New windows can also be created with the **New Window** command of the **Windows** menu. To delete a window, click its close box, or use the **Delete Window** command of the **Windows** menu. 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 **Adjust Position** subcommands of the **Windows** menu. The **Tile Horizontally** subcommand adjusts the windows so that they are full-width, but just tall enough to fill the screen, one above the other. The **Tile Vertically** subcommand adjusts the windows so that they are full-height, but just wide enough to fill the screen, one next to the other. The **Cascade** subcommand adjusts the windows so that they are all the same size and overlap each other uniformly from the upper-left to the lower-right.

Splitting Editing Windows

Each editing window can be split into multiple subwindows by using subcommands of the **Window Partitions** command. The **Split** subcommand divides the current window in half, either vertically or horizontally. When splitting windows, the aspect ratio of the editing window and its contents are used to determine how the window is split.

Once the original is split, only one half is the current window, and its border is highlighted. The current window changes, however, whenever the cursor moves to another window and a command is issued. You can grab the border and slide it to adjust the location of the split.

The **Split** command can be used repeatedly to subdivide windows into halves, quarters, and so on. Once the initial split has been made, there is no longer a choice between horizontal and vertical splits: Each division follows an alternating sequence to achieve a clean binary tree of windows.



To undo the partitioning of a window, use the **Delete** subcommand of the **Window Partitions** command of the **Windows** menu, which deletes the current partition and merges it with its neighbor. This command can be issued only to a window at the bottom of the subdivision tree. To back out completely from multiple partitions, use the **Make 1 Window** subcommand, which returns to a single partition.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 4: THE DISPLAY



4-4: Scaling and Panning



Scaling

The scale of a window's contents can be controlled in a number of ways. The **Zoom In** command of the **Windows** menu 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.

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

To examine a specific area of the display, use the **Focus on Highlighted** subcommand of the **Special Zoom** command, 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 *rectangle zoom* button, which allows you to drag–out a rectangle, and then zooms to that area. You can zoom to a specific area in a single step by using the **Highlight then Focus** subcommand, which lets you drag an area and then zooms to that area.

The **Make Grid Just Visible** subcommand of the **Special Zoom** command zooms (in or out) until the grid is minimally visible. Any further zoom–out from this point will make the grid invisible. If the grid is not being displayed, it is turned on.

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.

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 Windows 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).

In addition to these methods, panning can also be done from menu commands. The **Left**, **Right**, **Up**, and **Down** commands of the **Windows** menu 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 subcommands of the **Panning Distance** command to change that amount. The **Small** subcommand causes subsequent shifts to be about 15% of the screen size. The **Medium** subcommand causes subsequent shifts to be about 30% of the screen size. The **Large** subcommand causes subsequent shifts to be about 60% of the screen size.

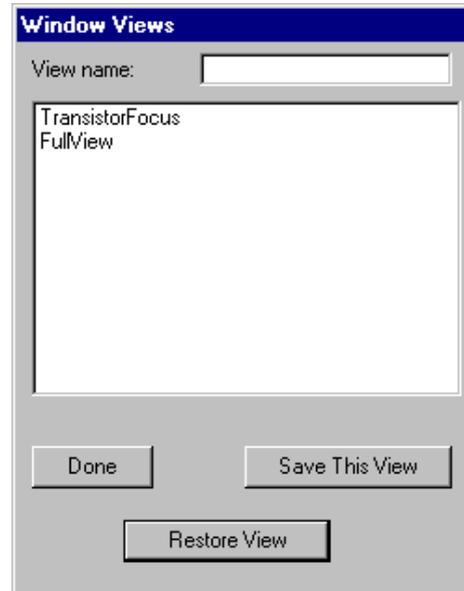


Another way to pan the window is to drag the cursor to the window edge. When the edge is hit, the window automatically pans by 10% of its size.

The **Center** subcommands of the **Windows** menu are rarely-used panning commands for shifting the window contents without scaling. There are two subcommands: **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 subcommand is useful only when bound to a keystroke, because you cannot issue the command and have a valid cursor location at the same time.

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 of the **Windows** menu 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.



[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 4: THE DISPLAY



4-5: Layer Visibility

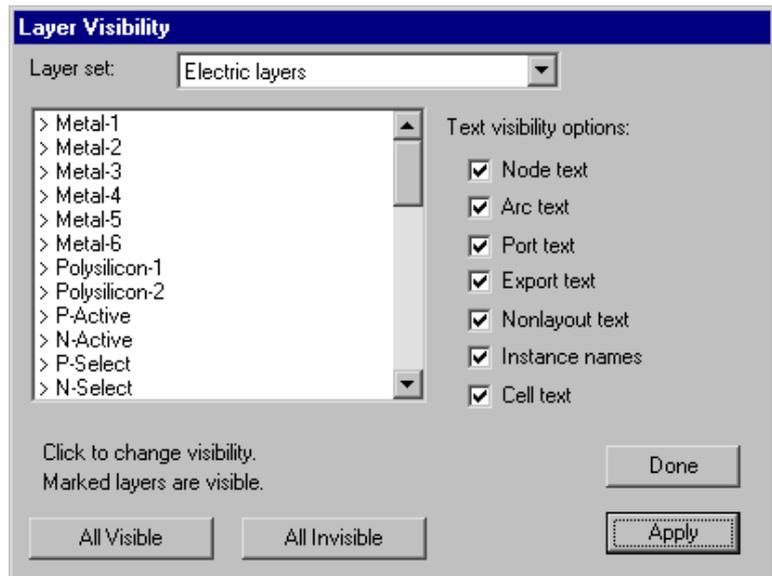


The nodes and arcs on the display are composed of more basic *layers*. By using the **Layer Visibility...** command of the **Windows** menu, you can control which layers are actually drawn.

A dialog is presented showing the layers in the current technology. By clicking on a layer entry, that layer's visibility is toggled (the ">" mark indicates that the layer is visible). Special buttons let you mark or unmark all of the layers.

This is a modeless dialog: it can remain up while other editing is being done. Therefore, you can apply visibility changes with the "Apply" button without dismissing the dialog.

Besides setting visibility by Electric layers, you can choose different layer sets which work with external layers (i.e. GDS layers). When the command finishes, the windows are redrawn.



The right side of the dialog 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](#). Note that this side is titled "Text visibility options" which means that these settings are saved, whereas those on the left side are not.

[Previous](#)

[Table of Contents](#)

[Next](#)



Chapter 4: THE DISPLAY



4-6: Colors



Electric's Color Model

The subcommands in the **Color Options** command of the **Windows** menu control the appearance of individual layers in the editing window. Before explaining these commands, it is useful to understand the distinction between *transparent* and *opaque* layers in Electric.

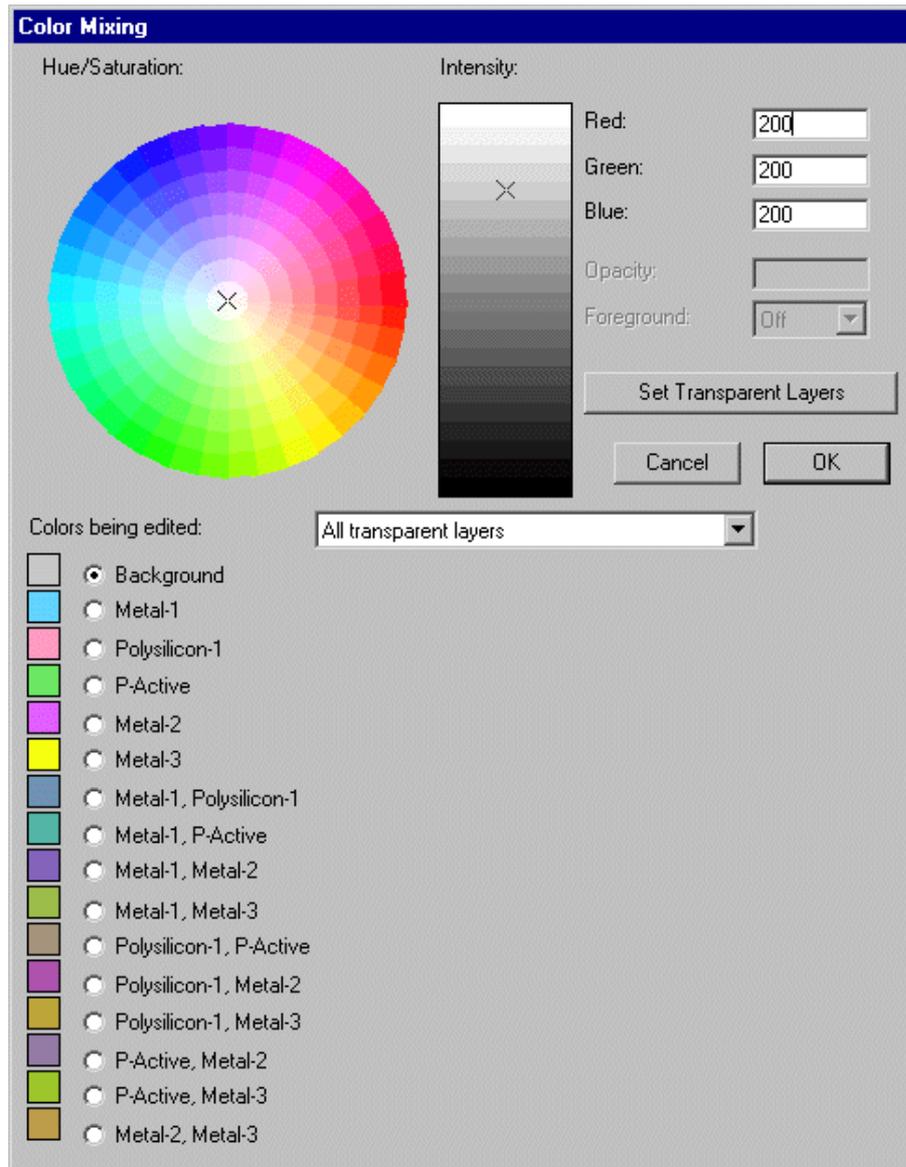
On color displays, up to five of the layers can be transparent, meaning that when they overlap each other, it is possible to see all of them. Typically, the most commonly used layers are transparent because it is both faster to draw and clearer to distinguish. The remaining layers in a technology are 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 stipple patterns are slower to draw, and because opaque color does obscure, the less common layers are drawn in this style. When editing colors, the opaque layers have only one color, whereas the transparent layers have up to 16 different colors, considering their interaction with other transparent layers.

Editing Colors

The first **Color Options** subcommand is **Edit Colors...**, which presents a color mixing dialog. The bottom half of the dialog shows a set of 16 colors that can be edited. A popup menu allows different sets of 16 to be selected. The choices are:

- "All transparent layers" (the 5 transparent layers and all combinations of two)
- "Special colors" (used for borders, text, grid, and other parts of the display)
- The five transparent layers (in Mosis CMOS, the default technology, these layers are Metal-1, Polysilicon-1, P-Active, Metal-2, and Metal-3)
- Banks of printable colors (these are used by the PostScript output system when "Color Merged" is selected in the **Print Options...** command of the **File** menu).





To edit a color, choose a particular color button and edit its value in using controls in the top half of the dialog. The mixing palette consists of a hue/saturation wheel and an intensity slider. You can also type numeric values for the Red, Green, and Blue. For printable colors, there is also an Opacity (1.0 if opaque, smaller values to control blending with other colors) and a Foreground factor ("On" if this layer can merge with others behind it).

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).

The "Set Transparent layers" button helps with the tedious task of defining all combinations of transparent colors. Because there are 5 transparent layers, it is necessary to define 32 different combinations of these colors. Instead, you can simply select the 5 transparent colors and the background color and then use this button to compute the remaining 26 combinations.



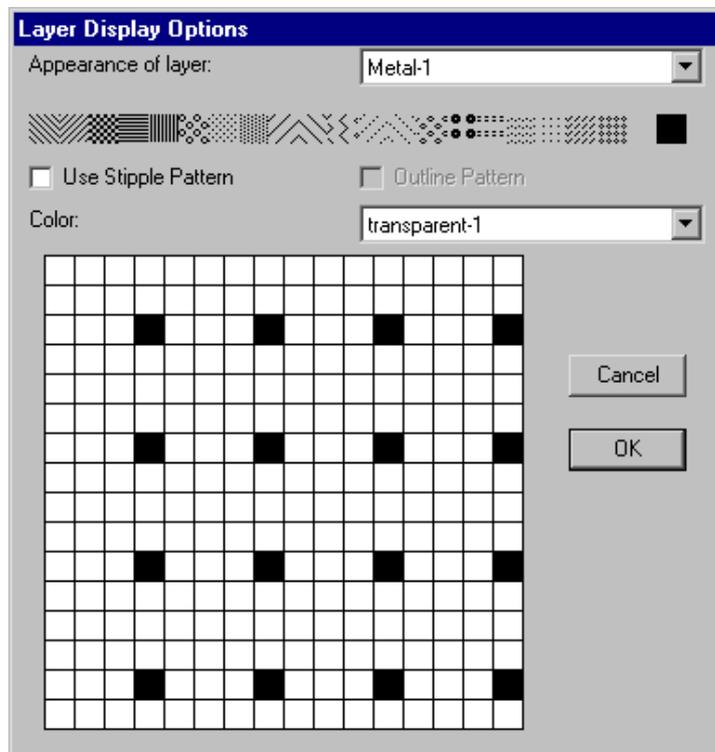
Another command for color manipulation is **Highlight Layer...**, which allows you to highlight a single layer. The command changes the colors so that all other layers are dimmer (actually 20% less saturated). The selected layer is thus highlighted on the display. Choose the "None" button to restore default colors. This command works as you click in the dialog, so that you can easily preview the circuit, one layer at a time. It only works on the transparent layers.



The final color subcommands are **Restore Default Colors**, which resets all colors to the default set for the current technology, **Black Background Colors** which resets all colors to the default for the current technology but with a black background, and **White Background Colors** which resets all colors to the default for the current technology but with a white background.

Setting the Color and Pattern of Layers

Besides editing colors, it is also possible to assign them to layers and to edit each layer's stipple patterns. The **Layer Display Options...** command of the **Windows** menu gives you this facility. Although many layers are drawn with solid colors on a color display, they all have patterns that are used for hardcopy output. The dialog that appears with this command allows the patterns to be edited, provides a set of predefined patterns, and allows you to determine whether or not the pattern is used on the display. You also have the option of outlining stippled polygons with a solid line. Finally, you can assign any transparent or opaque color to a layer.



[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 4: THE DISPLAY



4-7: Grids and Alignment



Drawing a Grid

The **Toggle Grid** command of the **Windows** menu 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 lambda unit apart. The term *lambda* indicates a generic spacing unit that can be scaled independently of the actual layout. For example, in the MOSIS CMOS technology, the value of lambda is 0.2 microns, as shown in the status area under the heading "LAMBDA". When the grid is displayed, the dots are therefore 0.2 microns apart. For more information on lambda, [Section 7-2](#).

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 location of the grid dots is always aligned with the "grab point" of the cell. Because this is normally the lower-left corner, the dots will always pass through that point. If, however, a Cell-Center node (named "Facet-Center" for historical reasons) is placed in the cell (with the **Cell Center** subcommand of the **New Special Object** command of the **Edit** menu) then the grid dots will always pass through that point.

The **Grid Options...** command 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.

	Horizontal:	Vertical:
Grid dot spacing (current window):	1	1
Default grid spacing (new windows):	1	1
Distance between bold dots:	10	10

Align grid with circuitry

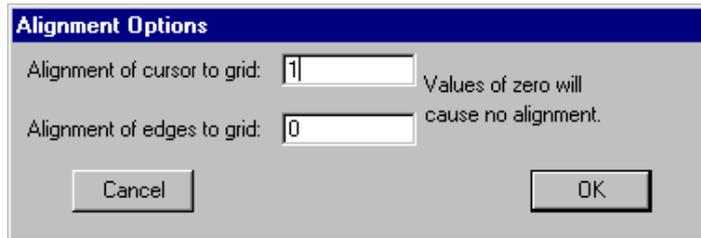
It is possible to change the horizontal and vertical grid dot spacings. You can also change number of grid dots between bold ones. Finally, you can choose whether or not to align the grid with the circuitry. When aligned, the dots are drawn so that they always fall on the "grab point" (normally the lower-left corner of the circuitry). When not aligned, the dots are drawn in the same location, regardless of the circuitry.



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

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).



The Alignment

Options... command presents a dialog in which alignment values may be set. For example, if the grid spacing is 2×3 , and the alignment is 0.5, then there are up to six different positions for placement inside a displayed grid rectangle.

A special grid setting is the alignment of edges, which initially is set to zero (no alignment). This alignment affects the edges of nodes and arcs (as opposed to their lower-left corner, where the grab-point resides). For example, if a 3 lambda wide wire is drawn into a 4×4 contact, the default will be to center that wire, which will place the edges of the wire on half-lambda grid locations. If, however, the edge alignment is set to 1 lambda, then that wire will be forced to one side of the contact so that its edges align. Note that it is not always possible to align edges properly, so you should always check your geometry if you insist on this feature.

The **Align to Grid** subcommand of the **Move** command of the **Edit** menu 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.

Aligning to Objects

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

The subcommand **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 subcommand **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 subcommand **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 subcommand **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.

Measuring

As an aid to precise alignment, the **Show Cursor Coordinates** command of the **Info** menu causes the X and Y cursor positions to be continuously displayed in the status area. This information appears where the Technology and Lambda used to be. To restore display of Technology and Lambda values, uncheck this



menu entry.

If you wish to find the distance between any two points on the display, use the **Measure Distance** command of the **Info** menu. After this command is issued, click in the circuit to set the "starting point". Then click repeatedly in the circuit to define the "ending point" and see the measured distance. To end distance measurement, issue the command again. The measured distance can be used by the **Array...** command to specify spacing (see [Section 6-4](#)).

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



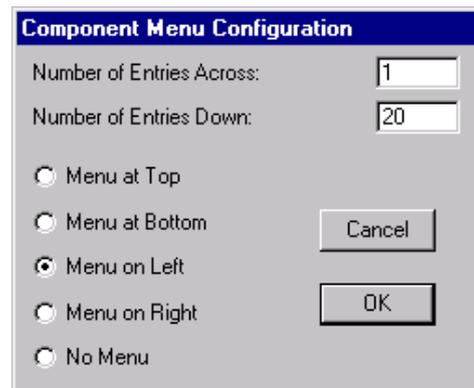
Chapter 4: THE DISPLAY



4-8: The Component Menu



Besides the standard pulldown menus, there is a *component menu* on the left side of the editing window. This menu can be controlled with the **Component Menu...** command of the **Windows** menu. Besides changing the location of the menu, you can change the number of entries it contains.



In general, it is not necessary to set the number of entries, because any change of technologies will automatically redraw the component menu with the correct number of entries.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 4: THE DISPLAY



4-9: Hardcopy



To make a paper copy of the contents of the current window, use the **Print...** command of the **File** menu. On UNIX, this command produces a temporary PostScript file, which is then spooled to the printer.

On the Macintosh and Windows, an image of the selected circuitry can be obtained with the **Copy** command of the **Edit** menu. This command copies the highlighted circuitry to the clipboard, which can then be pasted into most drawing applications and word processors.

On Windows, the resolution of printing and copying can be controlled by setting the "Print and Copy resolution factor" in the **Print Options...** dialog shown below. Changing this factor from 1 to 2 causes twice the detail in each axis, and an image that is 4 times larger.

As an alternative to printing, you can request PostScript or HPGL files. These files describe the circuitry graphically, and can be printed or inserted in other documents. To get PostScript, use the **PostScript** subcommand of the **Export** command of the **File** menu. To get HPGL, use the **HPGL** subcommand.

The **Print Options...** command of the **File** menu provides a number of options for generating print files. 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. Note that when printing the highlighted area, a precise selection can be made with the *rectangle select* button. For both PostScript and Macintosh printing, the "Plot Date In Corner" option causes additional information to appear in the corner of the plot. On UNIX systems, you can choose the printer



to use.

There are many PostScript options.

- The "Encapsulated" checkbox causes the PostScript output to be insertable in other documents. For encapsulated PostScript, it is also possible to specify, for each cell, a scale to use.
- There are four 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); "Color Stippled" which uses color stipple patterns for better overlap; and "Color Merged" which precomputes the layer overlap to produce a better-looking plot, but takes more time.
- 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 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. Checking the "Synchronize to file" checkbox 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.

For HPGL, you can choose the version that you want to generate (HPGL or HPGL/2). If you choose HPGL/2, you can specify the scale of the plot.

 [Previous](#)  [Table of Contents](#) [Next](#) 



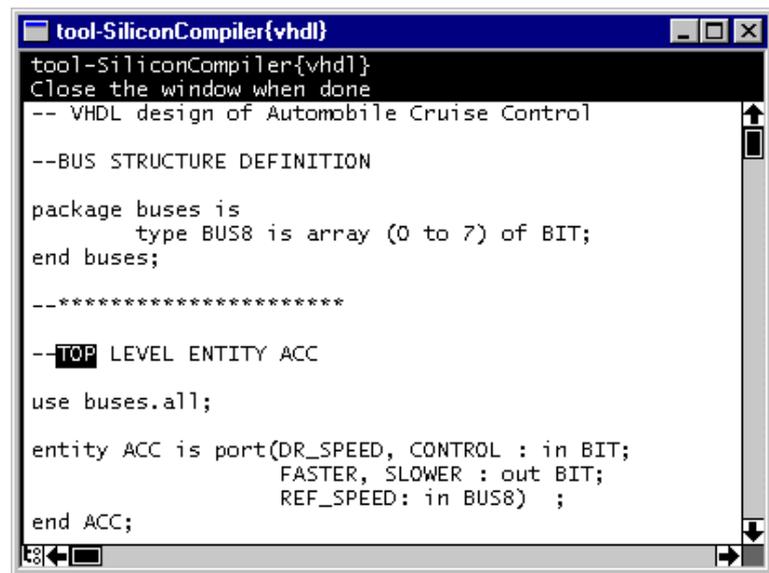
Chapter 4: THE DISPLAY



4-10: Text Windows



Certain commands cause text to appear in a window or window partition. This happens when a text-only view of a cell is edited (VHDL, Verilog, Netlists, or Documentation), when SPICE model cards are edited, or when design rules are being modified in the technology editor.



```
tool-SiliconCompiler{vhd}
Close the window when done
-- VHDL design of Automobile Cruise Control

--BUS STRUCTURE DEFINITION

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

--*****

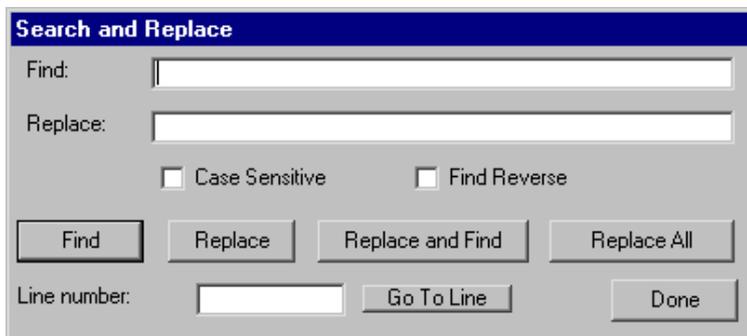
--TOP LEVEL ENTITY ACC

use buses.all;

entity ACC is port(DR_SPEED, CONTROL : in BIT;
                  FASTER, SLOWER : out BIT;
                  REF_SPEED: in BUS8) ;
end ACC;
```

There are two styles of text editing that can be used in these text windows. The default is a traditional "point-and-click" style which lets you select text with the cursor and replace/insert by typing. You can cut-and-paste with the **Cut**, **Copy**, and **Paste** commands of the **Edit** menu.

The other style of text editing is modeled after the EMACS text editor, and can be selected with the **Text Options...** command of the **Windows** menu. The EMACS-like text editor has only the basic commands and does not support macros, mini-buffers, split windows, or any advanced features. It exists only for those who demand the style of EMACS, even at a minimal level.



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



Note the **Find Text...** command can also be used to find text in a circuit.

If the text window is a text-only view of a cell, and if there are other layout views of the cell also on the screen, then you can make associations between the two. Just select the name of the network in the text window and use the **Show Network** subcommand of the **Network** command of the **Tools** menu.

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).

The contents of a text window can be saved to disk with the **Write Text Cell...** command of the **Cells** menu to save just the text. Use **Read Text Cell...** to read a file into a text window.

 [Previous](#)

 [Table of Contents](#)

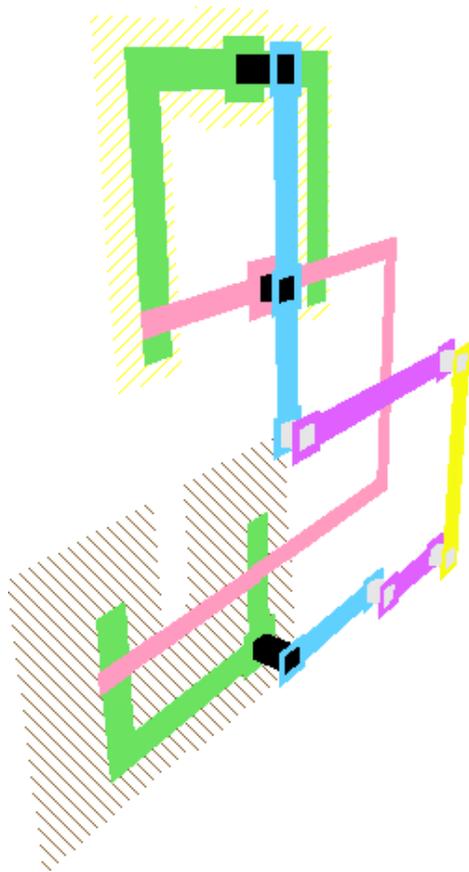
[Next](#) 



Chapter 4: THE DISPLAY



4-11: 3D Display



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

To see a circuit in 3D, use the **View in 3 Dimensions** subcommand of the **3D Display** command of the **Windows** menu. The circuit is displayed in 3D, and mouse movements cause the circuit to rotate.

For optimal 3D viewing, the circuit should be centered in the window before issuing the **View in 3 Dimensions** command. Also, the window should not be partitioned (split).

Beside rotation, you can pan, zoom, and twist the 3D display. To switch to these options, use the **Pan View Point**, **Zoom View Point**, and **Twist View Point** subcommands (and use **Rotate View Point** to rotate). These four display modes can also be invoked by simply typing the letters "p", "z", "t", and "r".

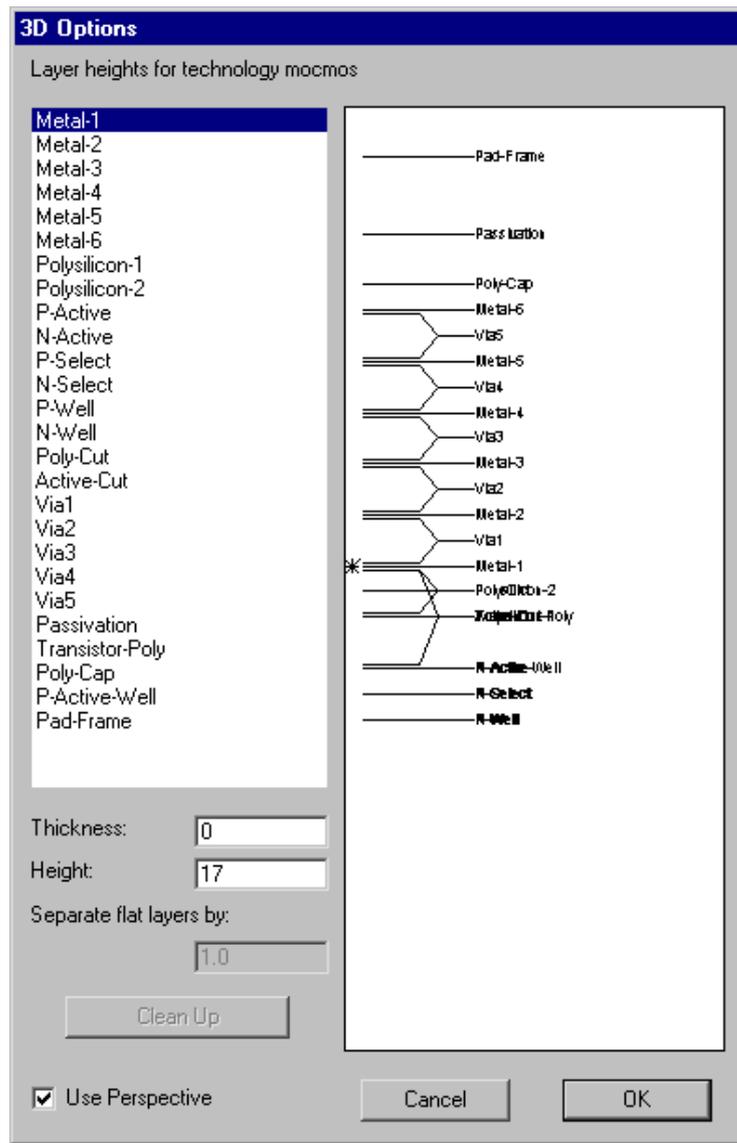
To return to a 2–dimensional view, use the **View in 2 Dimensions** subcommand.



To control the 3D view, use the **3D Options...** subcommand of the **3D Display** command of the **Windows** menu.

On the left side of this dialog is a list of layers in the current technology. On the right side is an edge-on view of the chip, showing which layers are above which others. 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 starred in the right-hand view.

Once selected, you can drag layers to the desired height (on the right side) or type a height value (on the left side). You can also set the thickness of a layer by typing a value into the field (typically, only contacts have thickness because they span between layers). Finally, you can choose whether or not to display with perspective. The "Clean Up" button and flat-layer separation items are not implemented at this time.



[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



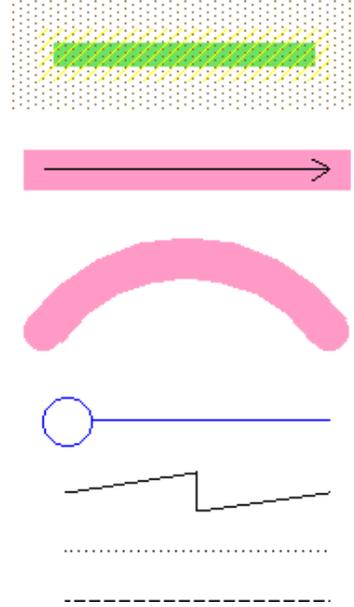
Chapter 5: WIRE PROPERTIES



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. In layout, they can be curved, directional, and more.



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.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 5: WIRE PROPERTIES



5-2: Constraints



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.

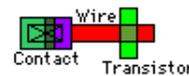
Rigid and Fixed-Angle Arcs

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.

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** subcommand of the **Cleanup Cell** command of the **Edit** menu.

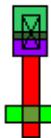
Original Structure



Contact rotated, unconstrained arc



Contact rotated, rigid arc

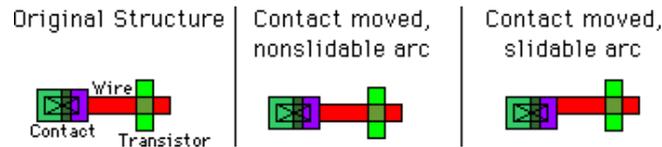


Contact rotated, fixed-angle arc



Slidable Arcs

The third constraint, also considered 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 in 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.



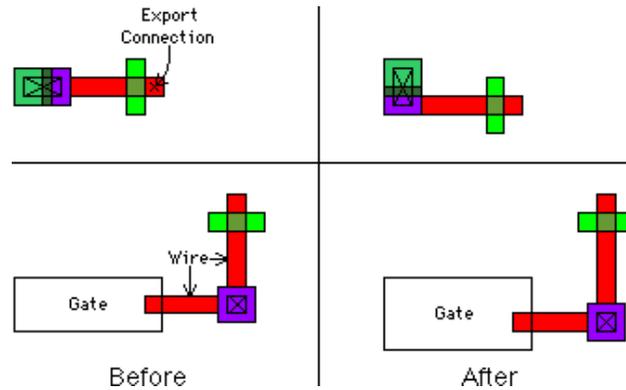
Slidable arcs may adjust themselves within the port area rather than move. For example, if a node's motion is such that the arc can slide without moving, then no change occurs to the arc or to the other node. Without the slidable constraint, the arc moves to stay connected at the same location within 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** subcommand of the **Cleanup Cell** command of the **Edit** menu.



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.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 5: WIRE PROPERTIES



5-3: Setting Constraints



The two most common constraints, rigid and fixed-angle, can be controlled from the **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 other constraints, a single arc must be selected and the **Get Info** command issued from the **Info** menu.

In the bottom-right corner of the arc information dialog are four check boxes that control constraints. This is the only way to affect the slidable constraint (which is not very commonly used). Also settable in this dialog is the temporary-rigidity constraint which indicates that an arc will act rigid or nonrigid for the next change only, after which it will revert to its previous constraint state.

Arc Information	
Type:	Polysilicon-1
Network:	a2
Name:	<input type="text"/>
Width:	<input type="text" value="2"/> Bus size: N/A
Angle:	180 <input checked="" type="checkbox"/> Easy to Select
Head:	Polysilicon-1-Pin
At:	(-210.5,13) <input type="button" value="See"/> <input type="button" value="Info"/> <input type="button" value="OK"/>
Tail:	P-Transistor
At:	(-193.5,13) <input type="button" value="See"/> <input type="button" value="Info"/> <input type="button" value="Attributes"/>
<input type="checkbox"/> Negated	<input type="checkbox"/> Ignore head <input type="checkbox"/> Rigid
<input type="checkbox"/> Directional	<input type="checkbox"/> Ignore tail <input type="checkbox"/> Temporary
<input checked="" type="checkbox"/> Ends extend	<input type="checkbox"/> Reverse head and tail <input checked="" type="checkbox"/> Fixed-angle
	<input checked="" type="checkbox"/> Slidable

[Previous](#)

[Table of Contents](#)

[Next](#)



Chapter 5: WIRE PROPERTIES



5-4: Other Arc Properties



In addition to the constraining properties, there are a set of characteristics that can be placed on arcs. These properties affect the appearance of the arc and cause it to be redrawn in a different style.

Directionality



For documentation purposes, it is possible to display a *directional* arrow on arcs that indicates flow. This property can be set or reset with the **Directional** command of the **Arc** menu. It may also be controlled by checking the appropriate box in the **Get Info** command of the **Info** menu.

The arrow always runs from the tail of the arc to the head of the arc unless you reverse the arc (with the **Reverse** command of the **Arc** menu or by checking the "Reverse head and tail" box in the **Get Info** dialog). If the head of the arc is "skipped" (with the **Skip Head** command or by checking the appropriate box in the **Get Info** dialog), then the arrow head is not drawn.

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 set or reset with the **Negated** command of the **Arc** menu. It may also be controlled by checking the appropriate box in the **Get Info** dialog.

Although Electric attempts to place the bubble on the correct end of the arc, it can be reversed with the **Reverse** command of the **Arc** menu or by checking the appropriate box in the **Get Info** dialog. Note that there can be only one negating bubble on an arc, so users who want two must use two arcs, connected by a pin.

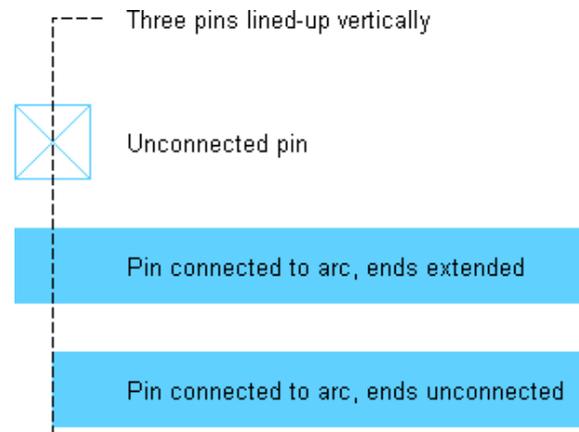
If the tail of the arc is "skipped" (with the **Skip Tail** command or by checking the appropriate box in the **Get Info** dialog), then the negating bubble is not drawn. Beware of doing this, because the negation is still part of the arc, even though it is not visible. Negated arcs make no sense in layout technologies and are ignored.



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 **Ends-extend** command of the **Arc** menu. It may also be controlled by checking the appropriate box in the **Get Info** dialog.

Individual ends of an arc may be controlled with the **Skip Head** and **Skip Tail** commands, or by checking the "Ignore head" and "Ignore tail" boxes in the **Get Info** dialog. When an end has been ignored, the property associated with it does not occur. This means that directional arcs have their arrowhead omitted (when the head is ignored); negated arcs do not have the bubble drawn (when the tail is ignored); and arcs with their ends not extended have that end extended.



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 **Get Info** 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.

Curvature

The final arc properties to be mentioned are used only in circular geometry. Although most arcs cannot handle curvature, those in the Artwork and Round CMOS ("rcmos") technologies can.



The **Curve through Cursor** command of the **Arc** menu 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. Click at the appropriate point on the screen after issuing these commands and the arc will change.

In the **Curve about Cursor** command, two curvatures are possible (clockwise and counterclockwise), so use the "Reverse head and tail" checkbox in the **Get Info** dialog to make a distinction. The **Remove Curvature** command makes the arc straight.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 5: WIRE PROPERTIES

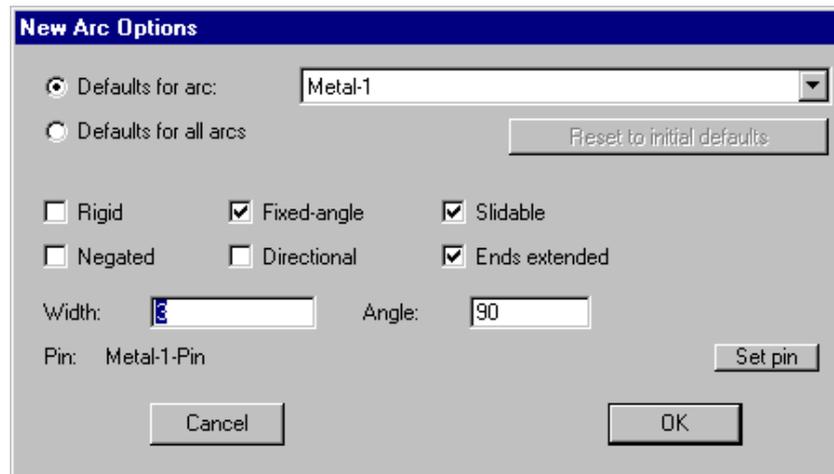


5-5: Default Arc Properties



The **Get Info** dialog and all of the commands in the **Arc** menu affect existing arcs, specifically those currently highlighted. To affect future arcs as they are created, use the **New Arc Options...** command of the **Arc** menu.

The dialog that appears allows you to set defaults for specific types of arcs or for all subsequently created arcs. When setting defaults for specific arcs, select the top radio button and then choose the arc from the list.



When setting defaults for all arcs, use the bottom radio button. When setting defaults for all arcs, you can use the "Reset to initial defaults" button to restore initial settings. Note that the default unit for typed values is *lambda*, unless another unit is explicitly mentioned (for more on lambda, see [Section 7-2](#)).

The rigid, fixed-angle, and slidable constraints may be controlled with this dialog, as can the negated, directional, and end-extension properties. It is also possible to specify the default width and angle for newly created arcs. The angle field is the preferred angle increment in degrees for newly created arcs (90 for Manhattan, 45 for Manhattan plus 45-degree angles, and 0 for any angle). The "Pin" is the default node to use when joining two of these arcs.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



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 of the **Edit** menu 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 Node Options...** command of the **Edit** menu) then the duplicated objects are placed immediately without dragging. The advantage to this mode is that the location used to place these objects is intelligently determined by any previous duplications.

If any of the nodes have exports on them, they are not duplicated (unless "Duplicate/Array/Extract copies exports" is set in the **New Node Options...** command of the **Edit** menu).

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 **Get Info** command of the **Info** menu) and then duplicated, the new ones will have different names (specifically, the old names with numbers appended).

Cut-and-Paste

Another way to make copies of nodes and arcs is with the cut-and-paste commands. The **Copy** and **Cut** commands of the **Edit** menu 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.

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 [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**.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 6: ADVANCED EDITING



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.

The **New Node Options...** command of the **Edit** menu provides a dialog for changing this information on subsequently created nodes. You can change the default size of any primitive node in the current technology by choosing the node and changing the values. Note that the default unit for typed values is *lambda*, unless another unit is explicitly mentioned (for more on lambda, see [Section 7-2](#)). You can change the default orientation of these primitives by checking "Override default orientation" and then entering the new rotation and transposition.

New Node Options

For primitive: Metal-1-Pin

X size of new primitives: 3

Y size of new primitives: 3

Override default orientation

Rotation of new nodes: Transposed

For all nodes:

Rotation of new nodes: 0 Transposed

Disallow modification of locked primitives

Move after Duplicate

Duplicate/Array/Extract copies exports

Node naming:

Length of facet abbreviations: 8

Primitive function abbreviations:

- unknown (node)
- pin (pin)
- contact (contact)
- pure-layer-node (plnode)
- connection (conn)
- nMOS-transistor (nmos)
- DMOS-transistor (dmos)
- pMOS-transistor (pmos)

unknown

Cancel

OK



The options in the middle section of the dialog apply to all new nodes. You can specify a default orientation (rotation and transposition). For more on orientation, see [Section 2–6](#).

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, described in [Section 7–8](#)), 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 of the **Edit** menu will place a copy automatically, without allowing the new location to be specified by the cursor. This has the advantage that it learns the location to use from past actions, so repeated **Duplicate** commands can be used to automatically place things regularly.

The check box "Duplicate/Array/Extract copies exports" requests that all node-copying operations also copy their exports. This includes the **Duplicate** and **Array** commands of the **Edit** menu and the **Extract Cell Instance** command of the **Cells** menu. By default, exports are not copied with their nodes.

The bottom section of the dialog allows you to specify node names to be used for the different types of nodes. These names are used when automatically naming nodes during netlisting. They are also used by the **Name All In Cell** and **Name All In Library** subcommands of the **Special Function** command of the **Edit** menu, which apply names to nodes and arcs.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 6: ADVANCED EDITING



6-3: Options



There are many commands in Electric that set options. These commands customize the system for your use. Here is a list of commands that set options, with references to them:

File menu:	Library Options, Copyright Options [3-9], CIF Options, GDS Options, EDIF Options, DEF Options, CDL Options, DXF Options, SUE Options [7-3] (IO Options submenu) Print Options [4-9]
Edit menu:	New Node Options [6-2] Selection Options (Selection submenu) [2-1]
Cell menu:	Cell Options [3-7]
Arc menu:	New Arc Options [5-5]
Export menu:	Port and Export Options [3-6]
View menu:	Frame Options [7-6] Icon Options [3-12]
Windows menu:	Grid Options [4-7] Alignment Options [4-7] Layer Visibility [4-5] Color Options (a submenu) [4-6] Layer Display Options [4-6] Text Options [6-8] 3D Options (3D Display submenu) [4-11] Messages Window Location [4-2]
Info menu:	General Options (User Interface submenu) [9-1] Quick Key Options (User Interface submenu) [1-9]
Technology menu:	Technology Options [7-1] Change Units [7-2]
Tools menu:	DRC Options, DRC Rules (DRC submenu) [9-2] Simulation Options (Simulation (Built-in) submenu) [10-2] Spice Options (Simulation (SPICE) submenu) [9-4] Verilog Options (Simulation (Verilog) submenu) [9-4] FastHenry Options (Simulation (Others) submenu) [9-4] Well Check Options (ERC submenu) [9-3] Antenna-Rules Options (ERC submenu) [9-3]



- NCC Control and Options (Network submenu) [9–6]
- Network Options (Network submenu) [6–9]
- Logical Effort Options (Logical Effort submenu) [9–12]
- Routing Options (Routing submenu) [9–5]
- VHDL Options (VHDL submenu) [9–10]
- Silicon Compiler Options (Silicon Compiler submenu) [9–9]
- Compaction Options (Compaction submenu) [9–11]
- Java Options [11–1]

These options are stored in a file called "electricoptions.elib" in your "lib" directory (on UNIX, the file is stored in "~/electricoptions.elib"). Deleting this file will cause your options to revert to the default state set by Electric.

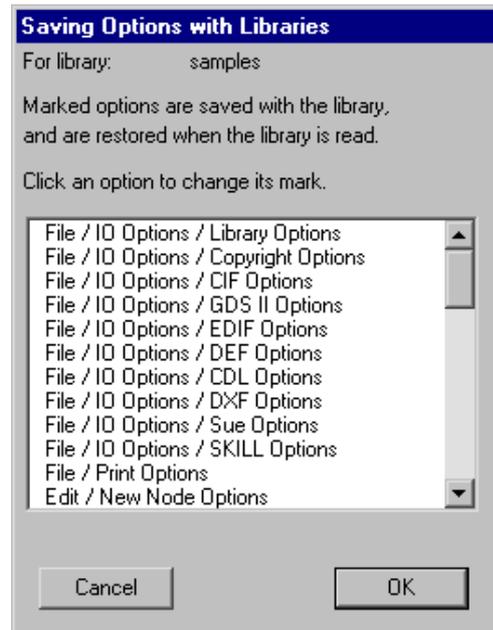
Options are saved when Electric exits. To save the options sooner, use the **Save Options Now** subcommand of the **Option Control** command of the **Info** menu.

To see which options are being saved, use the **Examine Saved Options...** command. This not only lists the options that are being carried between each session, it also shows which ones have been changed in this session.

To help find options, use the **Find Options...** subcommand of the **Option Control** command of the **Info** menu. This presents a list of all option commands and lets you search it by keyword.

There are times when you want to save options with a library. For example, a library of standard cells, designed for the Silicon Compiler, will want to store Silicon Compiler options in it so that the user of the library can have the proper options set.

To request that a set of options be saved with the current library, use the **Saving Options With Libraries...** subcommand of the **Option Control** command of the **Info** menu. Select the options that you want saved and then save the library. When this library is read in, options in that library will override the current settings.



[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 6: ADVANCED EDITING



6-4: Making Arrays



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

The **Array...** command of the **Edit** menu takes the currently highlighted objects and replicates them many times. You specify the replication in the X and Y directions and the geometry is replicated.

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 are centered, then the original circuitry will be placed in the middle of the array instead of the corner.

There are four ways to specify spacing: *edge overlap*, *centerline distance*, *characteristic spacing*, or by *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](#)). 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 (use the **Measure Distance** command of the **Info** menu to set this amount). Note that the default unit for typed values is *lambda*, unless another unit is explicitly mentioned (see [Section 7-2](#)).

Array Current Objects

X repeat factor: Flip alternate columns
 Stagger alternate columns
 Center about original

Y repeat factor: Flip alternate rows
 Stagger alternate rows
 Center about original

X edge overlap: Space by edge overlap
 Space by centerline distance
Y edge overlap: Space by characteristic spacing
 Space by last measured distance

Linear diagonal array
 Generate array indices
 Only place entries that are DRC correct



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 **Get Info** command of the **Info** menu 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.

 [Previous](#)  [Table of Contents](#) [Next](#) 



Chapter 6: ADVANCED EDITING



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...** subcommand of the **Move** command of the **Edit** menu.

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.



Negative spread distances compact the circuit. Note that the default unit for typed values is *lambda*, unless another unit is explicitly mentioned (see [Section 7-2](#)).

Another way to spread or squeeze circuitry is to use the compaction tool (see [Section 9-11](#)).

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



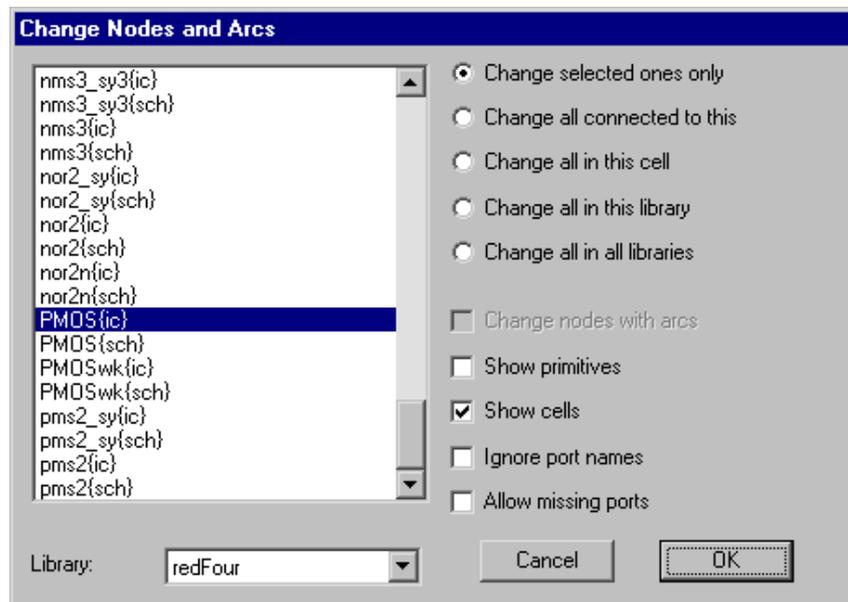
Chapter 6: ADVANCED EDITING



6-6: Replacing Circuitry



The **Change...** command of the **Edit** menu 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.



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.

Special Considerations

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. However, once changed, you can use the **Get Info** command of the **Info** menu to set the parameter and turn it into an Electrolytic Capacitor. Besides Capacitors, parameters can be found on Diodes, Transistors, Sources, and Two-Ports (the four-connection primitives such as VCCS).

Because arcs can connect only to certain types of nodes, it can be difficult to replace both. For example, if you wish to convert all Metal-1 into Metal-2, then you must replace all Metal-1 arcs with Metal-2 arcs, *and* you must replace all Metal-1-Pins with Metal-2-Pins. It is not possible to do this all at once, nor is it possible to do it in any two-step fashion (you cannot replace the arcs, because the pins won't connect to the new type, and you cannot replace the pins, because the arcs won't connect to the new type).

The solution is to use the Universal arc and the Universal Pin (from the Generic technology, see [Section 7-9](#)). These components can connect to any other, and so they can be used as intermediate placeholders. The solution to converting Metal-1 into Metal-2 is as follows:

- Replace all Metal-1 arcs with Universal arcs.
- Replace all Metal-1-Pins with Metal-2-Pins.
- Replace all Universal arcs with Metal-2 arcs.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



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 of the **Edit** menu 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.

Electric stores only the last 20 changes, so anything older than that cannot be undone.

In Electric, almost every command is undoable. This includes some commands that you would not normally consider undoable, such as the mouse click which changes highlighting. Although most commands are undoable, 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 or technology deletion) are not undoable. Finally, pure informational commands (as found in the **Info** menu) are not undoable.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 6: ADVANCED EDITING



6–8: Text



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 **Get Info**) so that they have text on them.
- 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 the **Text (nonlayout)** subcommand of the **New Special Object** command of the **Edit** menu; the **Add Verilog Code/Declaration** subcommands of the **Simulation (Verilog)** command of the **Tools** menu; the **Add SPICE Card** subcommands of the **Simulation (SPICE)** command of the **Tools** menu).

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.

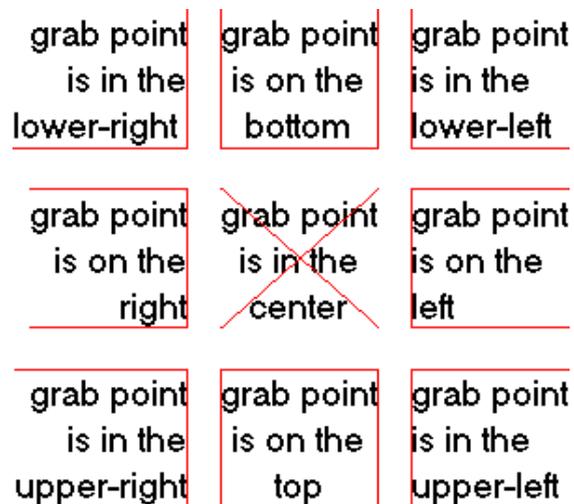


Selecting Text

The only category of text in the above list 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 pieces of text, the only way to select them is to use the *special select* button. By default, the name of an unexpanded cell instance requires this button. 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 Options...** subcommand of the **Selection** command of the **Edit** menu.

All text is attached to its node or arc at a *grab-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 grab-point. Typically, the highlighting consists of an "X" through the text. This indicates that the grab-point is in the center. If a "U" is drawn in any of four orientations, it indicates that the grab-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 grab-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*).

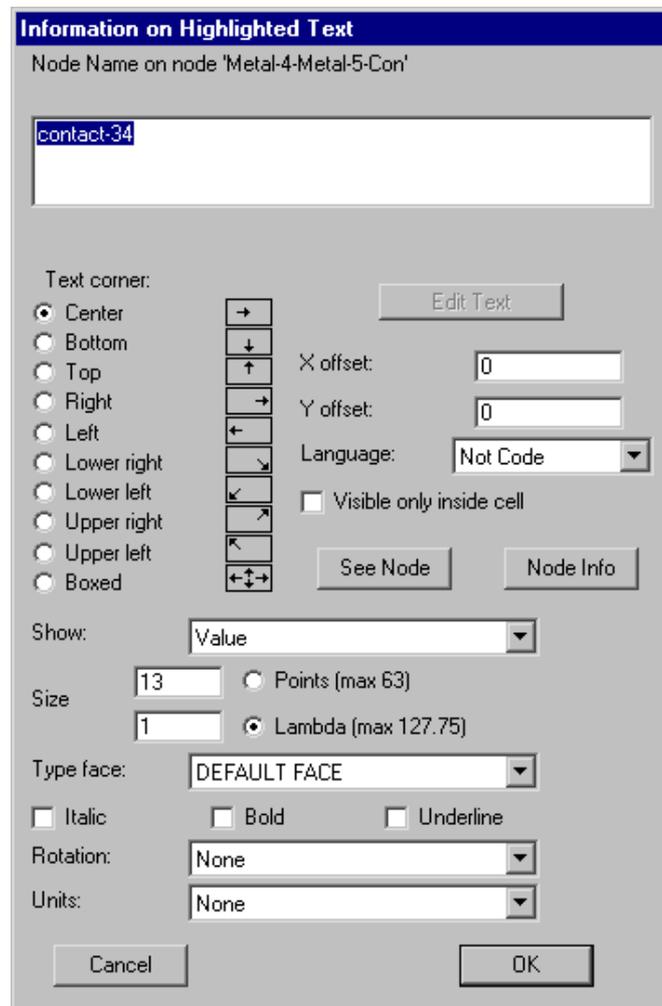


Modifying Text

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.

There are two ways to change the actual text: *in-place editing* or with a dialog. In-place editing is done by double-clicking on the text. After the double-click, all of the text is selected. Portions of the text can be selected by clicking over it. To insert or replace text, simply type. When done editing, click away from the text to end the editing mode.

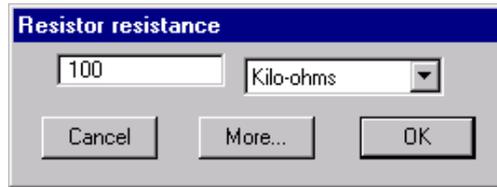
Dialog control of text is done by using the **Get Info** command of the **Info** menu. This dialog allows modification of the text, size, font, style, grab-point, rotation, and even the offset of the grab-point from the attached node or arc. Note that the offset is always relative to the center of the attached object. The size of text can be absolute (given in "points") or relative (given in lambda units). The font of text can use the default face 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](#) for more on these units. The "Language" option allows the text to be code in an interpretive language, in which case, the evaluation of that code is displayed. (see [Section 11-1](#) for more on languages). You can choose to show the text value, the name of the piece of text, or both. The little arrows next to the grab-point options show where the point of attachment lies on the text.



If the text contains more than 1 line, then the only way to change it is to click on the "Edit Text" button, which closes the dialog and enters in-place editing mode. The "See Arc" button highlights the arc on which the text is attached (the button is "See Node" if the text is on a node). Similarly, the "Arc Info" (or "Node Info") button brings up the **Get Info** dialog for that object. The checkbox "Visible only inside cell" requests that the text not be drawn when an instance of the cell is examined. In addition, for text objects (those created

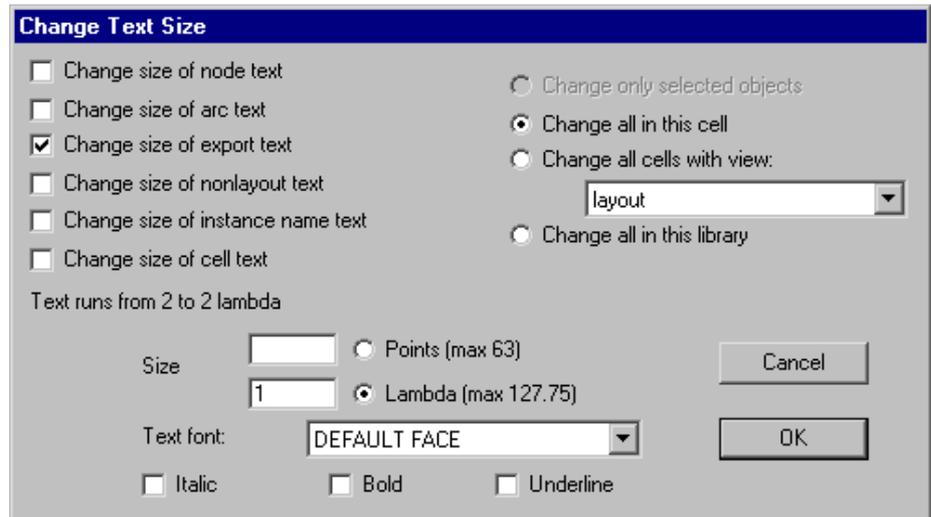


with the **Text (nonlayout)** subcommand of the **New Special Object** command of the **Edit** menu) the location of the text will not affect the bounding box of the cell. This means that the text can be placed arbitrarily far outside of the actual layout, and it will not affect the hierarchy.



For special pieces of text that the system understands, you may get a customized dialog when you double-click. For example, if you double-click over the resistance value of a resistor, a special dialog will appear to set the resistance. To change other information, use the "More..." button to see the general dialog.

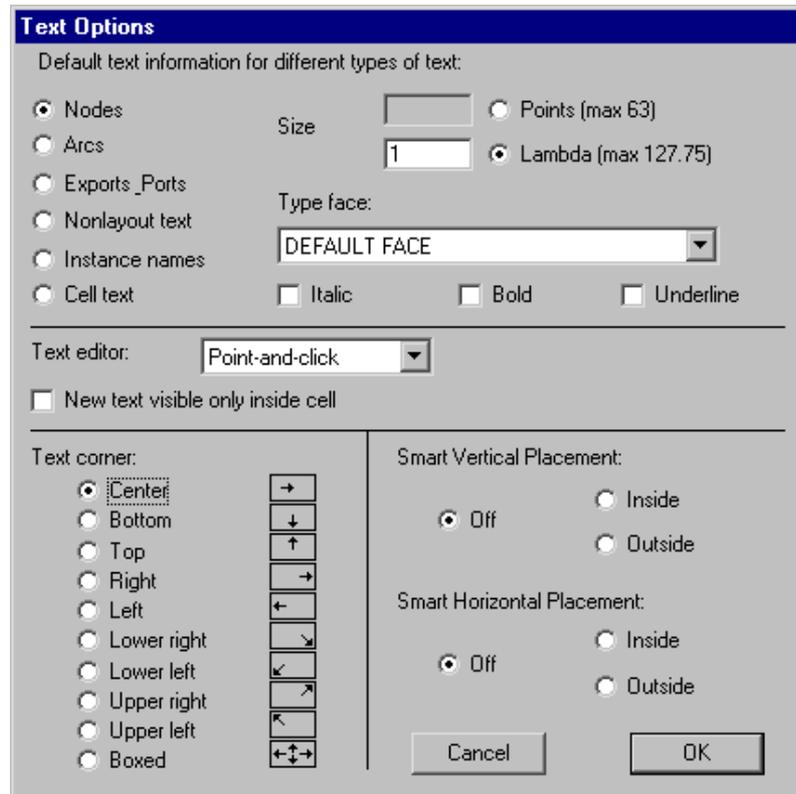
The **Change Text Size...** subcommand of the **Special Function** command of the **Edit** menu allows you to change the size, face, and style of any text object. You can choose which of the 6 classes of text you wish to change, and you can choose whether to make the changes only on selected objects,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



Text Defaults

To change the default size and grab-point of all new text, use the **Text Options...** command of the **Windows** menu. The top part of the dialog lets you set the default size, face, and style of text that will appear in different locations (on cell instances, on nonlayout text, on exports, on nodes, and on arcs). Nonlayout text is the freestanding text that is created with the **Text (nonlayout)** subcommand of the **New Special Object** command of the **Edit** menu.

The center section of the dialog has two unrelated controls. The field labeled "Text editor" allows you to choose which style of text editing to use when working with large amounts of text (see [Section 4-10](#) for more on text windows). The "New text visible only inside cell" checkbox requests that nonlayout text objects be drawn only when inside of the cell, and not when instances of the cell are expanded.

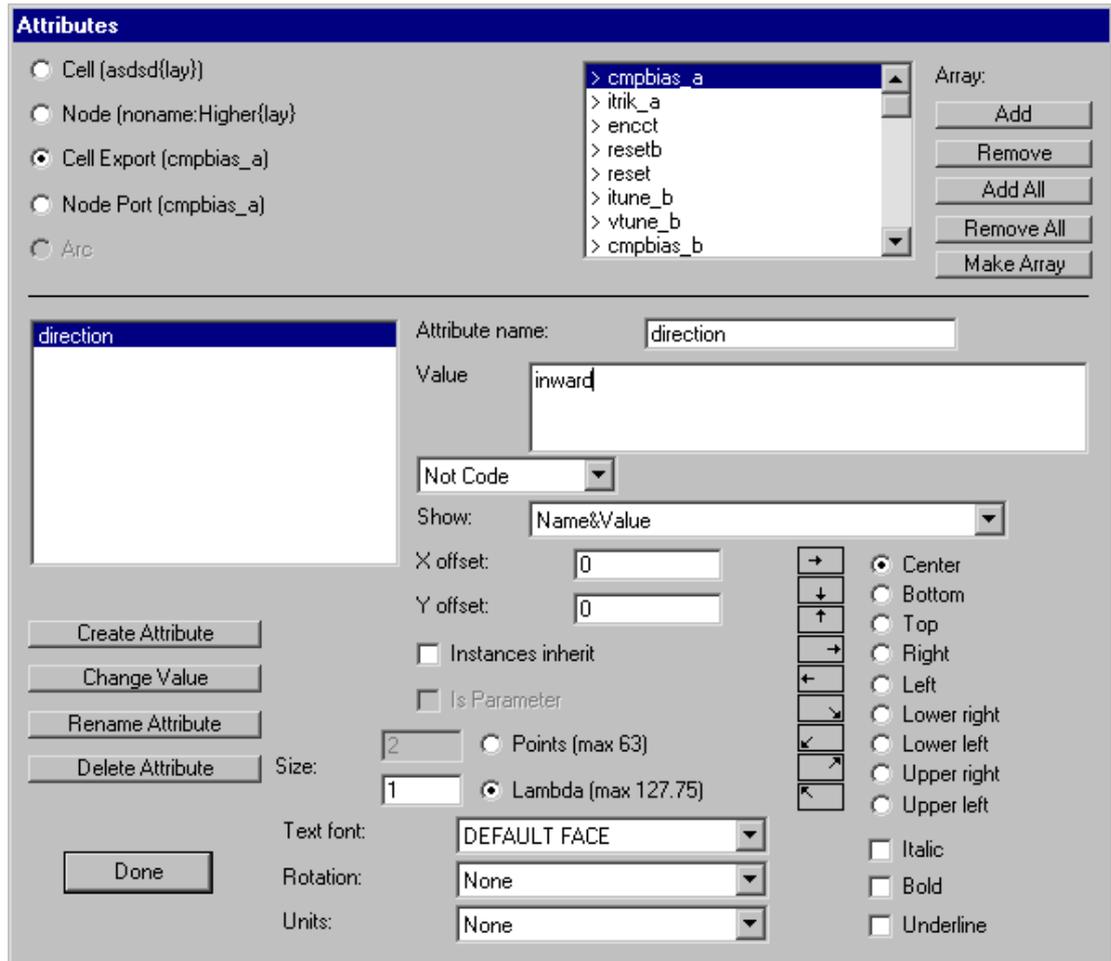


The lower-left part of the dialog lets you set the grab-point of subsequently created text. The lower-right part of the dialog controls "smart placement" of text, which adjusts the grab point according to the environment of the text. This currently applies only to export names, which are placed relative to the arc connecting to the exported node. For example, if a node on the left end of a wire is has an export, and the "Smart 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.



Text Attributes

You can place arbitrary text attributes on nearly any part of the circuit by using the **Define...** subcommand of the **Attributes** command of the **Info** menu.



Attributes can be placed on these objects (selected in the upper-left):

- The current cell.
- The currently selected node.
- Any of the exports in the current cell (select the particular export from the list in the upper-right).
- Any of the ports on the current selected node (select the particular port from the list in the upper-right).
- The currently selected arc.

The list of attributes is shown on the left. You can create a new attribute by typing its name in the "Attribute name:" field and its value in the "Value:" field and then clicking the "Create Attribute" button. You can delete an attribute with the "Delete Attribute" button. A selected attribute can have its value changed by typing a new value and clicking the "Change Value" button. An attribute's name can be changed with the "Rename Attribute" button.

Just below the attribute's value is a field that reads "Not Code". 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 LISP code, then the LISP interpreter will be invoked, and the attribute



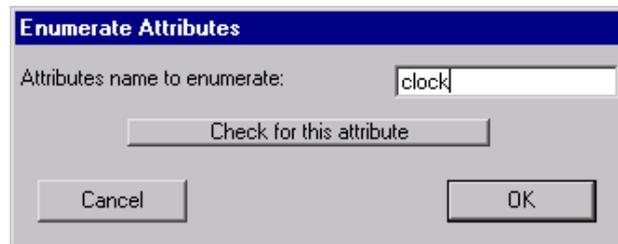
will actually be "8". For more on interpretive languages, see [Section 11-1](#).

You can change the type of unit by using the popup menu on the bottom (choices are capacitance, resistance, inductance, current, voltage, or distance). See section [Section 7-2](#) for more on these units.

For attributes on cells or exports, you can request that they be inheritable with the "Instances inherit" checkbox. When this is checked, newly created instances of the cell will have copies of this attribute on them. Using this scheme, an attribute can be considered to be a *parameter*, where values set on the instances are used inside of the cell. The "Is Parameter" checkbox should be selected in this case (note that the proper way to create parameters is with the **Cell Parameters...** subcommand, not this dialog). If you check "Visible only inside cell" in the attribute's **Get Info** dialog, then the inherited attributes will not be displayed.

It is often desirable to have attribute values that have unique names. If the value of an inherited 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.

Another way to create attributes with unique values is to place a "?" in the attribute value and then use the **Enumerate...** subcommand on the attribute name. This command finds all occurrences of that attribute, and replaces the "?" with a unique numeric value.



If you create a new inheritable attribute and wish to propagate it to existing instances, select the instance and use the **Update Inheritance** subcommand of the **Attributes** command of the **Info** menu. To propagate attributes to all instances, use **Update Inheritance All Libraries**.

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, value, and inherited value be displayed. When an attribute is shown, additional information is relevant. The grab-point of the text can be chosen from the list in the lower-right. The X and Y offset of the text from the attached object can be specified. The size of the text can be specified in relative or absolute units. The font of the text can be chosen from the popup list. The style of the text can be any combination of Italic, Bold, or Underline. You can even specify the orientation of the text, in 90-degree increments.

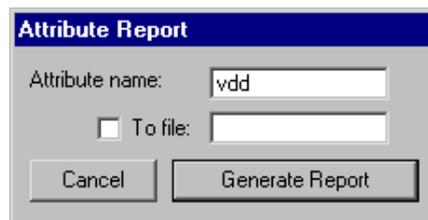
When there are too many visible attributes, the display can become cluttered. Use the **Layer Visibility** command of the **Windows** menu to control the text. (attributes on nodes are controlled by the "Node Text" checkbox; those on arcs with the "Arc Text" checkbox, etc.)

Special buttons exist in the upper-right for applying changes to many ports or exports. By using the "Add", "Remove", "Add All", or "Remove All" buttons, you can select a subset of names (those with a ">" are selected). Then, by using the "Make Array" button, the currently selected attribute is copied to all selected locations.

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



To help organize the attributes in a circuit, the **Attribute Report...** subcommand of the **Attributes** command of the **Info** menu lists every occurrence of a particular attribute name. You can also request that this command dump the report to a disk file.



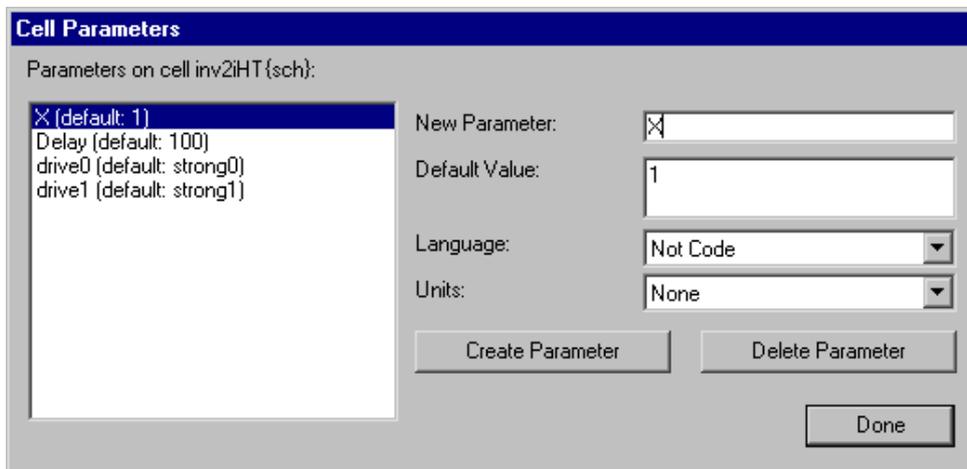
Cell Parameters

Parameters are special types of attributes that communicate information down the hierarchy, from a cell instance to its contents. 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](#)).

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 7-5](#)). 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 (see [Section 11-1](#) for more on interpretive languages).

Parameters are created with the **Cell Parameters...** subcommand of the **Attributes** command of the **Info** menu.

The dialog shows the existing parameters, and allows you to create and delete them. Each parameter has a default value that will be used if no instance value can be found. You can also set the type of electrical unit that this parameter describes (capacitance, cesistance, etc.) See section [Section 7-2](#) for more on these units.



Cell parameters are implemented as inheritable attributes. Inheritable attributes are automatically placed on each newly-created node instance (thus, they are inherited from the prototype to the instance). Inside of the cell, the parameter is shown with its name, default value, and actual value from up the hierarchy. For example, the parameter defined in the above dialog will appear in the cell as the string "Strength=?;def=2". This means that the actual value from up the hierarchy is not known ("?") but the default is 2.



When an instance of this cell is created, a new attribute will be placed on it with the default value. In this example, the instance will have the text "Strength=2" on it. Since this attribute is separate from the defining one in the cell, you can edit and change its value. Besides clicking on the text to edit the value, you can also see all parameters on a node by selecting the "Parameters" button in the node's **Get Info** dialog. Once an instance is created with a parameter value, you can descend the hierarchy from that instance and see the actual parameter value inside of the cell. So, if you changed the instance text to read "Strength=15", then descending into the cell will show the string "Strength=15;def=2".

A parameter, defined in a cell, always appears on every instance of that cell. If you have added a parameter after creating instances, use the **Update Inheritance** or **Update Inheritance All Libraries** commands.

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 make changes to the parameter locations on the sample icon, you can propagate those changes to icon instances with the commands **Update Locations** (updates only the selected icon instances) or **Update Locations All Libraries** (updates all icon instances).

If you do not wish to see a parameter's text on any instance, select the parameter text inside of the cell, use **Get Info** from the **Info** menu, and check "Visible only inside cell". To override the visibility of cell parameters on an instance, use the **See All Parameters on Node**, **Hide All Parameters on Node**, and **Default Parameter Visibility** subcommands of the **Attributes** command of the **Info** menu. These commands make all parameters visible, invisible, or back to their default as specified in the cell.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 6: ADVANCED EDITING



6–9: 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** subcommand of the **Network** command of the **Tools** menu. This will highlight all arcs on the currently selected networks. If the design is very dense, you can select one or more networks by name with the **Select Network...** subcommand of the **Selection** command of the **Edit** menu.

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 Options...** subcommand of the **Network** command of the **Tools** menu and check "Ignore Resistors". Then highlighted networks will pass through them. See section [Section 7–6](#) for more on resistors.

There are many commands that can be used to get information about the networks in a cell:

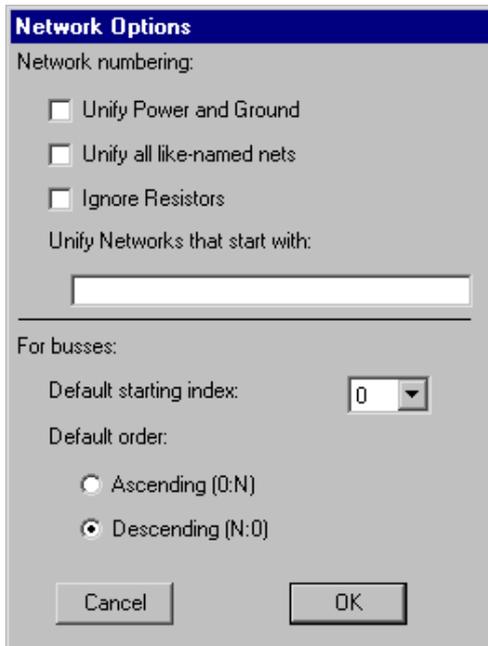
- The **List Networks** command of the **Info** menu shows a list of the named nets in the current cell.
- The **List Exports on Network** command of the **Info** menu lists all export names on the currently highlighted network. This list contains the names of exports at all levels of the hierarchy, 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.
- The **List Exports below Network** command of the **Info** menu 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.
- The **List Connections on Network** command of the **Info** menu 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, and 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.
- The **List Geometry on Network** command of the **Info** menu lists all geometry in the current cell that is connected to the current network. This reports the area and perimeter of all attached layers.



Naming Networks

Network names are derived from export names and arcs that are named in a cell. The name given to an export is the network name for all arcs connected to that export. Similarly, the name given to an arc (by setting the name field in the **Get Info** command) becomes the name of the network for all connected arcs. You can rename a network by changing the specific export or arc, or by using the **Rename Network...** command of the **Info** menu.

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."



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. To cause this same effect in all views (such as "layout"), use the **Network Options...** subcommand of the **Network** command of the **Tools** menu and check the "Unify all like-named nets" item.

You can also use this dialog to request that all networks that start with a particular set of letters should be unified. For example, if the "Unify Networks that start with" field is set to "clk", then the networks "clk01" and "clkExtra" will be merged into a single network. Multiple merges can be specified, separated by spaces. For example, if the field is "pwr gnd" then all nets starting with "pwr" will be merged, and all nets starting with "gnd" will be merged.

When busses must be automatically generated (during netlisting, for example) the bus indices can be set to start at 0 or 1, and they can ascend or descend.



Bus Naming

The Bus arc of the Schematics technology is a special arc that can carry multiple signals. When giving a network name to Bus arcs, it is possible to specify complex bus names. Bus names can be lists (for example, "clock,in1,out" which aggregates 3 signals into a 3-wide bus) or they can be arrays (for example, "A[0:7]" which defines an 8-wide bus). Arrays indexes can be individual values, or ranges of values (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]). Finally, 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 must give names to these unnamed busses. You can control the way that these busses are numbered by setting the "Default starting index" field in the **Network Options...** dialog. You can also select whether the numbering should go up or down.

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 Signals** subcommand of the **Network** command of the **Tools** menu 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.

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".

By default, 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.

Although this unification is the proper thing to do for schematics, it is not always proper for IC layout. For example, in MOS technologies, two ports exported with the "power" characteristic are not on the same net unless they are actually connected (there may be multiple power rails that do not connect). As a circuit debugging aid to ensure that power and ground networks are properly connected, Electric can be instructed to unify power and ground networks in ALL technologies, regardless of their actual connectivity. The **Network Options...** subcommand of the **Network** command of the **Tools** menu has the "Unify Power and Ground" item which causes all power and ground networks to be combined. This unification of all supply rails can be disabled by unchecking the menu entry. By highlighting power and ground networks with and without this option, designers can see whether all of their supply rails are fully connected.

The **Validate Power and Ground** subcommand of the **Network** command of the **Tools** menu 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.



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 specially 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.

The Global node is diamond-shaped, and it has a name and characteristics 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.

Not all netlisters in Electric use Global signals. At this time, only SPICE, Verilog, IRSIM, and the Network Consistency Checker make proper use of this feature.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 6: ADVANCED EDITING



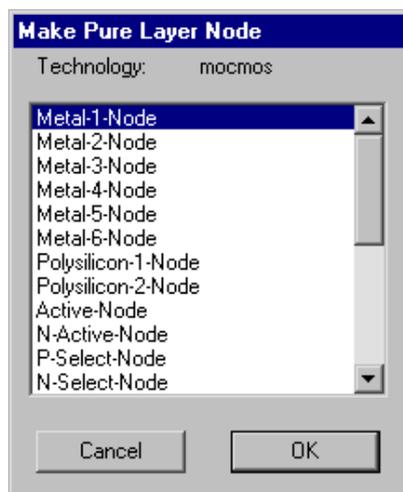
6-10: Outline Editing



What is an Outline?

For some primitive nodes, it is not enough to rotate, mirror, and scale. These primitives can be augmented with an *outline*, which is a polygonal description. The **Outline Edit** subcommand of the **Special Function** command of the **Edit** menu is used to create, delete, and move individual points in a node's outline.

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](#)). 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 7-7](#)).

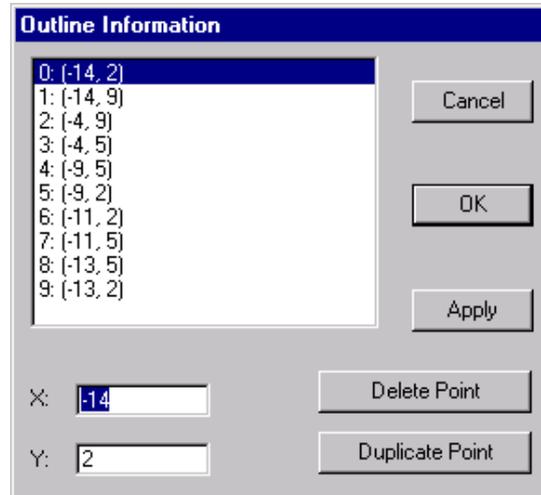


For arbitrary shapes on arbitrary layers, use the *pure-layer* nodes in the IC layout technologies. The pure-layer nodes are those in the **New Pure-Layer Node...** command of the **Edit** menu. For example, the node called "Metal-1-Node" in the CMOS technologies looks like a rectangle of the Metal-1,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



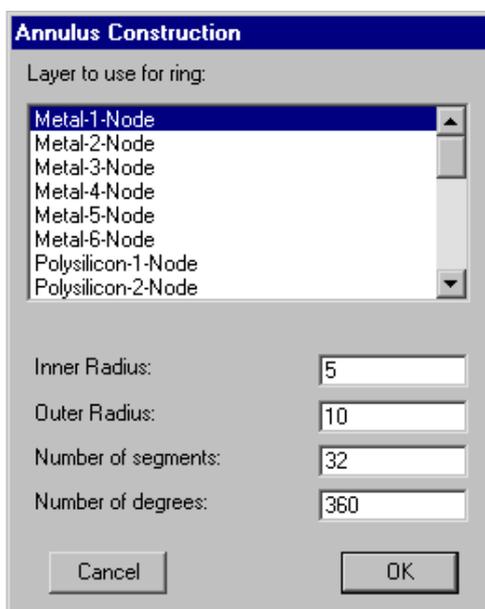
Manipulating Outlines

To manipulate outline information on the currently highlighted node, use the **Outline Edit** subcommand of the **Special Function** command of the **Edit** menu. After issuing this command, the window is highlighted with a blue border. In this mode, the *selection* button is used to select and move a point on the outline, and the *creation* button adds a new point after the selected one. You can also use the left and right arrow keys to move around the outline. The **Erase** command of the **Edit** menu changes to **Erase Point** and deletes the highlighted outline point (so does the Delete key). The **Rotate** and **Mirror** commands change to **Rotate about Point** and **Mirror about Point** which allow the entire node to pivot about the currently selected point. Finally, the **Get Info** command of the **Info** menu changes to **Get Outline Info** and displays a dialog which allows precise manipulation of the data.



When done editing the outline, use the **Exit Outline Edit** subcommand (in the same location as **Outline Edit** was in the **Special Function** command of the **Edit** menu).

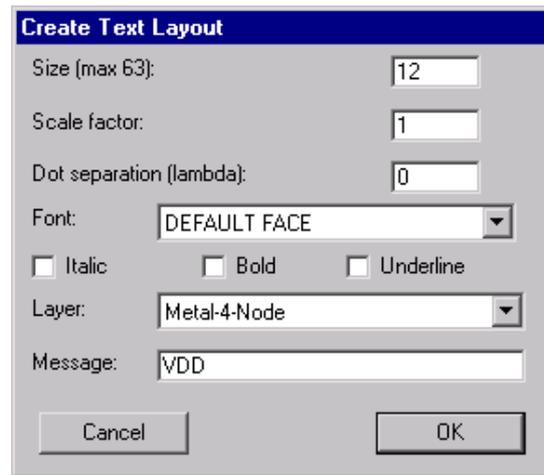
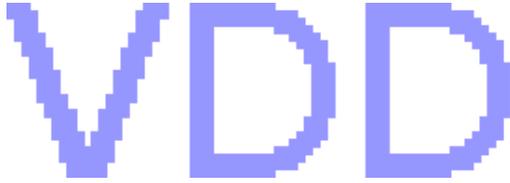
Special Outline Generation



To generate a doughnut shaped outline, use the **Annular Ring** subcommand of the **New Special Object** command of the **Edit** 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 outline, use the **Text (layout)...** subcommand of the **New Special Object** command of the **Edit** 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).



 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 6: ADVANCED EDITING



6-11: 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 *master library* in a shared location, and copies of that library on each user's local disk area. Users work on views of cells by checking them out of the master library, making changes, and then checking them back in. The project management system ensures that only one user can access a cell view 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. The project management system handles design history by creating a new version of a cell view each time it is checked out of the master library. The user's copy of the library contains only the most recent version of each cell view, taken from the master library. When a user updates their library from the master library, newer versions are brought in and substituted for older versions. Unless the user specifically asks for an older version, it is removed from the local library.

Because the project management system uses versions to manage design progress, users work with "cell views", rather than "cells". The terms are nearly equivalent, with the restriction that a "cell view" is the cell that is the most recent version. Throughout the manual, and even in this section, the term "cell" is used. But be aware that when you use the project management system, the term "cell" implies the most recent version only. Users should not make explicit use of older versions in their own design. For example, it is not appropriate for a user to use two different versions of a cell view explicitly, because they are considered to be part of a single cell view's history.

All commands to the project management system can be found under the **Project Management** command of the **Cells** menu. Subcommands exist there for checking cells in and out, updating local libraries from the master library, and controlling individual users.

Creating a new Project

The first step needed to use the project management system is to take a library file and convert it into a master library. The library disk file must already be located in a shared location which must be readable and writable by all users of the system. The library must then be read into Electric, and the **Build Project** subcommand used. This creates additional files in the shared location, specifically a project status file and a folder of individual cell libraries that mimics the hierarchical structure of the master library. At this point, the master library should be saved to that shared location.

Individual users can now begin to work on the library. They do this by copying the shared master library to their local space. All cells of this library are checked-in, and no work may be done until a cell is checked-out.



If the users have not already done so, they must create accounts in the project management system. The **Set User...** subcommand allows user name setting, and creation of new users. Each user must give a password. Note that the user database is kept in the library directory, which should also be in a shared location. Use the **Set Paths...** command of the **Info** menu to see where this directory resides.

Checking Cells In and Out

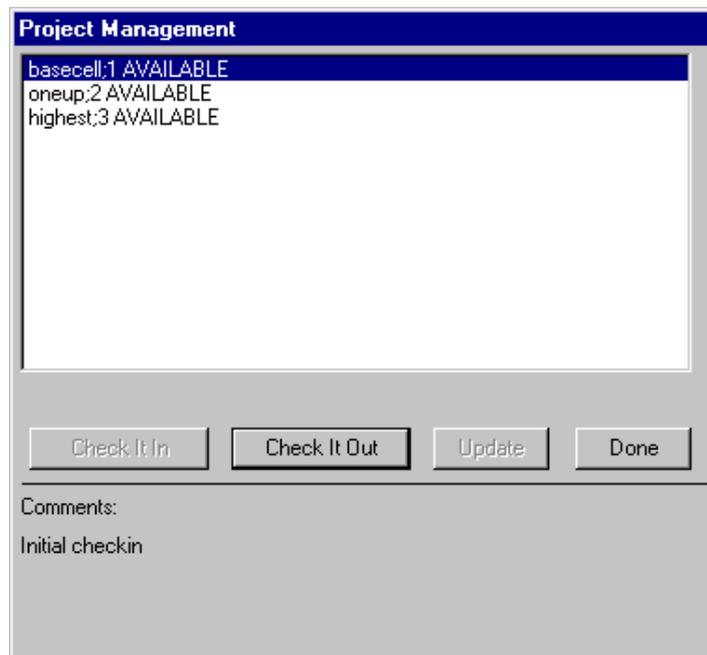
When a cell is not checked out, you cannot make changes to it. Any change is immediately undone by the project management system. This means that a change which affects unchecked-out cells, higher up the hierarchy, will also be disallowed.

To check-out the current cell, use the **Check Out This Cell** subcommand. You may be prompted for your user name and password if you haven't already given it. You can give a check-out message to help document the intended change. 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** subcommand. You will again be prompted for a documentation message about the change. No further changes will be allowed to the cell.

To update your library so that it contains the most recent versions of every cell, use the **Update** subcommand. You will be given a list of cells that were replaced.

Note that the check-out, check-in, and update functions are all combined into one subcommand, **Check In and Out**, which presents a dialog showing the state of all cells, and allows full control over the library.



If, in the course of design, a new cell is created, it must be added to the project management system so that others can share it. Use the **Add This Cell** subcommand to include the cell in the database. Conversely, if a cell is deleted, and should be removed from the project management system, use the **Delete This Cell** subcommand.



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 master library. When the cell is checked-in, that new version also goes into the master library. When updates are done, newer versions are obtained from the master library, and appropriate substitutions are performed.

One feature of this scheme is that you can get old versions of a cell, if you want to back-out of any changes. The subcommand **Old Version of This Cell** pulls an older version of the current cell into the library. This old version is available for editing and display. When done, the old version can be deleted.

[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 6: ADVANCED EDITING



6-12: Emergencies



Database Corruption

Electric is a vast system with many facilities. Although the individual pieces work correctly, it is possible, through improper usage or system fault, for the database to become corrupted. If Electric acts strangely, try the **Redo Network Numbering** subcommand of the **Network** command of the **Tools** menu to regenerate network information.

A more thorough check may be done with the **Check and Repair Libraries** command of the **Info** menu. Note that this command clears the history list so no undo can be made after the check. When in doubt, save your work first.

Running out of Memory

Another problem that can arise, particularly on a pre-System-10 Macintosh, is that Electric may run out of memory. As this situation approaches, you will be given warning messages, and finally you will be unable to work on new circuitry. Sadly, it is not generally possible to recover memory by deleting unwanted circuitry. The wise move is to save your library and exit Electric. Then make more memory available and run it again.

Crash Recovery

On some operating systems (Windows and UNIX/X11) Electric keeps a log of every keystroke and mouse action. If a crash occurs, you can replay the log and recreate the lost work.

The log file is called "electric.log" (on UNIX, it is called ".electric.log"). When Electric exits, it renames the file to "electriclast.log". If Electric starts up and finds a file called "electric.log", it presumes that this file is from a crash, and offers to replay it. You can force a log file to be replayed by using the **Playback Log File...** subcommand of the **User Interface** command of the **Info** menu.

When playing back a log file, cursor motion is used to advance the playback. Therefore, to playback the entire file, simply keep moving the cursor back and forth. At any time, you can abort the playback by clicking the mouse.

Be warned that the new log file that is created as a result of playing back an old log file is not necessarily correct.

Note also that in order to make a replayable log file, you cannot partially save your work. If, for example, you are editing two libraries and you save only one of them, then the state of the world cannot be recreated after a crash. Also, if you have changed options, then when you save your libraries, you should also save the options.



Electric will prompt you to save any unsaved information in order to keep the session logging current. If you choose to make only a partial save, session logging will be disabled.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 7: DESIGN ENVIRONMENTS



7-1: Technologies



Many Different 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, D.O.D., and even round geometry. A simple Bipolar technology is available. Even Gallium Arsenide technologies have been built into Electric, but because they made use of proprietary design-rules, they are not distributed with the system.

The library "samples.txt" contains a number of examples of the different technologies in Electric (you can read it with the **Readable Dump** subcommand of the **Import** command of the **File** menu). The table below lists the examples:

Cell	Technology	Description
tech-MOSISCMOS	mocmos	MOSIS CMOS rules
tech-RoundCMOS	rcmos	Experimental round CMOS rules
tech-nMOS	nmos	n-Channel MOS rules
tech-SchematicsDigital	schematics, digital	Digital schematics layout
tech-SchematicsAnalog	schematics, analog	Analog schematics layout
tech-PCB	pcb	Printed-circuit board layout
tech-DigitalFilter	efido	Digital filter architecture
tech-GEM	gem	Temporal logic specification
tech-Artwork	artwork	Graphical design

Electric makes no restrictions about mixing components from different technologies. While editing a cell, you can switch technologies and start using new components along side the ones from the former technology. It is up to the designer to ensure that the resulting circuit is sensible.

What is in a Technology

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 attributes, etc.

Within a technology, there are three classes of primitive nodes: *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, so there is one for every layer in the technology.



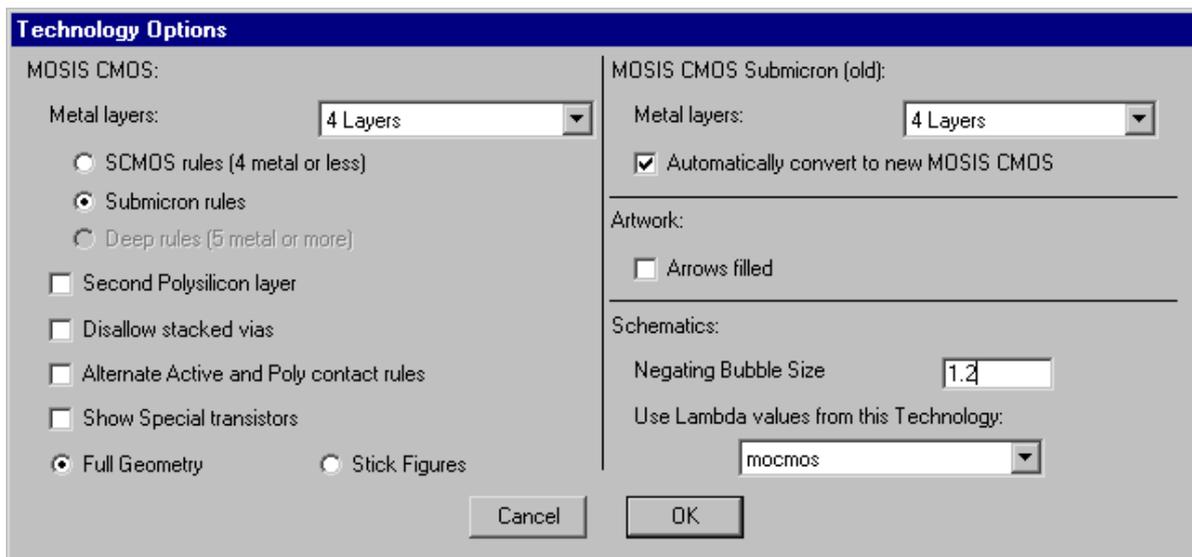
The component menu on the left side of the editing window contains arcs at the bottom (the menu entries with red borders), pin nodes above that (these appear as boxes with a cross inside), and components at the top (the more complex layer combinations). The pure-layer nodes are not in the components menu, but are available from the **New Pure-Layer Node** command of the **Edit** menu (see [Section 6-10](#)). Note that if you use the **Components Menu...** command of the **Windows** menu, and increase the number of menu entries, then the pure-layer nodes will be visible in the components menu.

Controlling Technologies

When Electric begins, the status area shows the current "Technology". To work with a different technology, use the **Change Current Technology...** command of the **Technology** menu. This will prompt you with a list of available technologies. Once a choice is made, the component menu on the left will be redrawn to show the primitive nodes and arcs in the new technology. Because each technology has its own transparent colors, you will also notice a chromatic change in the display.

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 **Cell Options...** command of the **Cells** menu and uncheck "Switch technology to match current cell".

To see a list of primitive nodes and arcs in the current technology, use the **Describe Current Technology** command of the **Technology** menu. To get a detailed list of layer, node, and arc information in a technology, use the **Document Technology** command.



Some technologies have settable options that further customize them. The **Technology Options...** command of the **Technology** menu provides a dialog for controlling those options. More information about this dialog is available from the individual technology sections on MOSIS CMOS ([Section 7-5](#)), Schematics ([Section 7-6](#)), and Artwork ([Section 7-7](#)).



Chapter 7: DESIGN ENVIRONMENTS

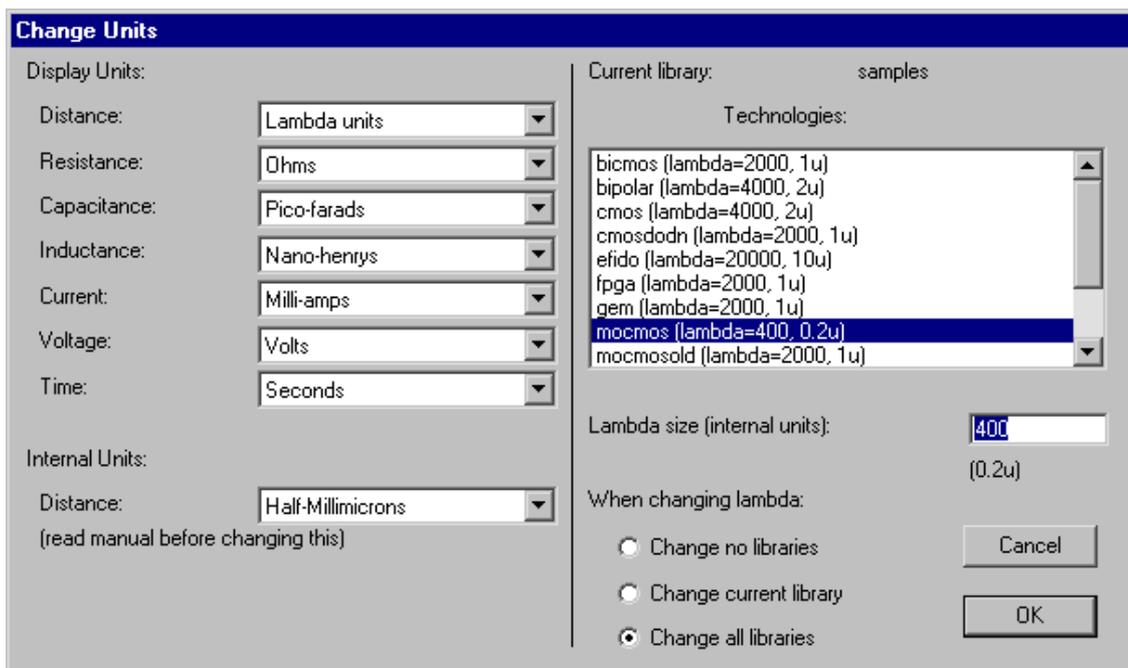


7-2: Units



Electric stores all geometry in terms of an *internal unit* of distance, but displays all distances in terms of a *display unit*. In addition, Electric uses a basic grid unit called *lambda*, which provides for scalable design.

To change the value of the internal unit, the display units, or the value of lambda, use the **Change Units...** command of the **Technology** menu.



The rest of this section describes the use of this dialog.

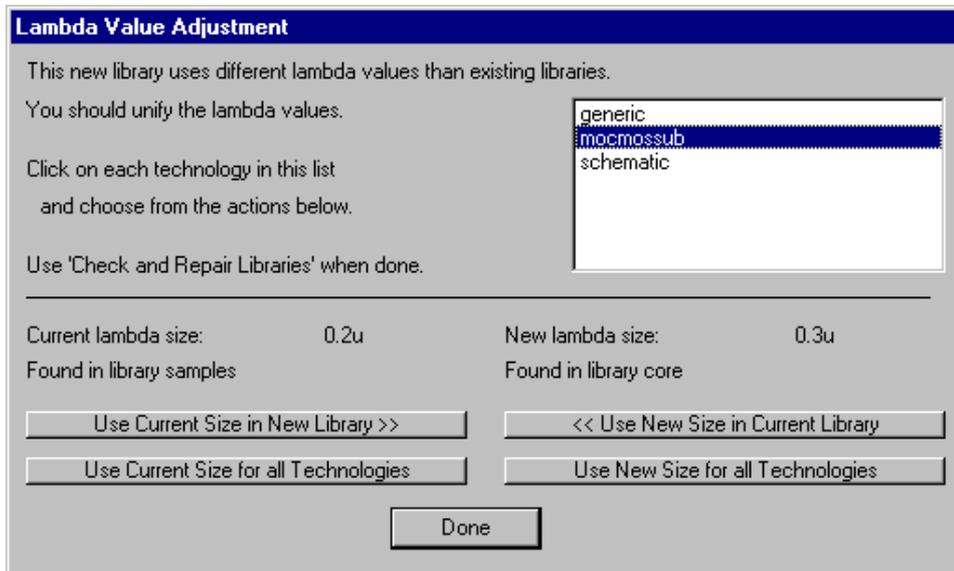
Lambda

The right side of the **Change Units...** is used to manipulate the value of Lambda. Lambda is the basic unit of design, expressed in internal units. For example, the MOSIS CMOS technology has transistors that are 2x3 lambda units in size, and a value of lambda that is 400. Because the internal unit is the half-nanometer (the half millimicron), 400 internal units is 0.2 microns, and the transistor is 0.4 by 0.6 microns in size. Note that the value of lambda is shown in the status area.

Each library has a list of lambda values for each technology. This means that two libraries can have different lambda values for the same technology, and this can cause trouble if the libraries reference each other.



Because of this, when a library is read in with lambda values that are different from existing libraries, you are presented with a dialog to help resolve these differences.



You can choose to scale the new library or the existing libraries so that they match. You can also choose to ignore the problem if you know that the inconsistent lambda values will not cause trouble.

If you use the **Change Units...** command and change the value of lambda, you have three choices for how to adjust the existing circuitry:

- "Change no libraries" causes the technology to scale, but existing objects are left alone. This may cause existing objects to change their aspect ratio, even if they don't change their physical size.
- "Change current library" causes the technology and all objects in the current library to scale. The objects and their cells will now be at a different size, but will appear the same as before.
- "Change all libraries" causes the technology and all objects in all libraries to scale. The objects and their cells will now be at a different size, but will appear the same as before.

For example, if you change lambda for the "mocmossusb" technology from 400 to 800, and you scale the libraries, then the existing 2x3 lambda transistor will be affected. Its area will now be 0.8 by 1.2 microns in size. Thus, lambda-based designs are scalable to the desired process.



Display Units

The upper-left side of the dialog is concerned with display units. Distance can be expressed in Lambda units (a scalable unit) or you can switch the display unit to a real distance. Changing the display units affects only how numbers are presented, not their internal storage. For example, selecting the "Microns" display unit causes Electric to describe distances in microns, rather than lambda units. When a real display unit is used, appropriate notation is used to express distances (i.e. a 3 micron wire is "3u" wide).

When you type a distance value in Electric, that value is presumed to be in the current display unit. However, if you add a specific unit symbol to the end of the value, those units are used. These unit symbols are recognized:

<u>Unit</u>	<u>Unit Symbol</u>	<u>Example</u>
Inches	"	3"
Centimeters	cm	7cm
Millimeters	mm	12mm
Mils	mil	4.5mil
Microns	u	-90u
Centimicrons	cu	100cu
Nanometers	nm	1nm

Besides distance units, you can also change the way that various other electrical units are displayed. Once again, changing these values does not affect the database in any way, only how the numbers are printed.

Internal Units

The lower-left side of the dialog is concerned with Electric's internal units. Electric has a choice of two different internal units. For IC design, the half-nanometer (half millimicron) is appropriate because it is small enough to represent chip geometries. However, because Electric uses 32-bit integers for storage, the half-nanometer can only express values from about -42 inches to +42 inches. If you are using Electric for larger designs (printed circuit boards or other large physical items) then this is too restrictive.

You can choose to switch the internal unit to the half-decimicron (50 nanometers), which is 100 times larger, and gives a much wider range of real distance. However, when you change the internal unit, all database units are scaled to keep their size constant. This means that if you switch to the half-decimicron, very tiny distances may get truncated. In general, the internal unit should not be modified unless you are sure you know what you are doing.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 7: DESIGN ENVIRONMENTS



7-3: I/O Specifications



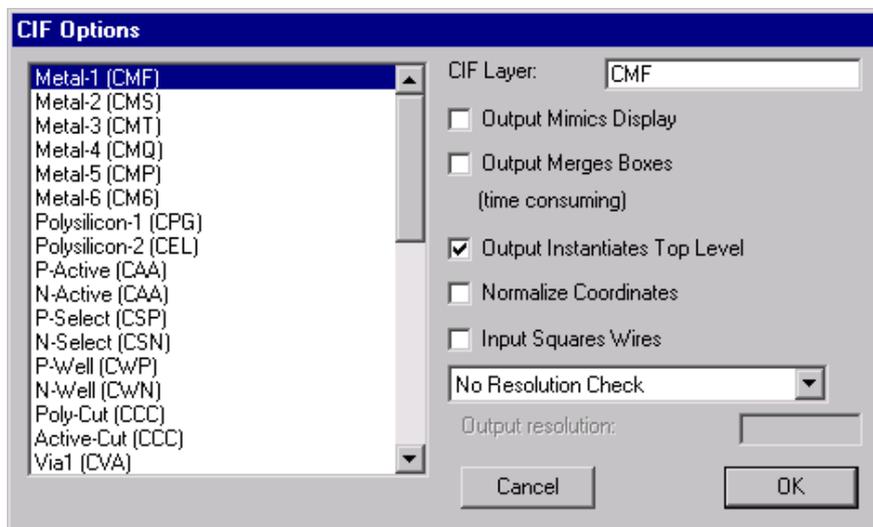
Electric is able to read and write circuits in a number of different formats. This is done with the **Import** and the **Export** commands of the **File** menu (see [Section 3-9](#)). To properly control translation, the **IO Options** command has many subcommands for the different file types.

Unfortunately, many of these formats are pure geometry with no information about the circuit connections. When read, the input formatter creates pure-layer nodes to describe the geometry. 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.

A partial solution to this problem exists now. The **Node Extract** subcommand of the **Network** command of the **Tools** menu will replace the pure-layer nodes in the current cell with connected nodes and arcs. It isn't guaranteed to work in all situations, and it doesn't recognize transistors at all. However, it starts the conversion process (which must be completed by hand). Also, the code is available and some enterprising programmer may want to finish (or rewrite) it.

CIF Control

CIF options are controlled with the **CIF Options...** subcommand of the **IO Options** command of the **File** menu.



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 into the dialog.



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. To revert to the default state, uncheck the item.

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.

The "Normalize Coordinates" option causes the coordinate system of each CIF cell to be centered at zero.

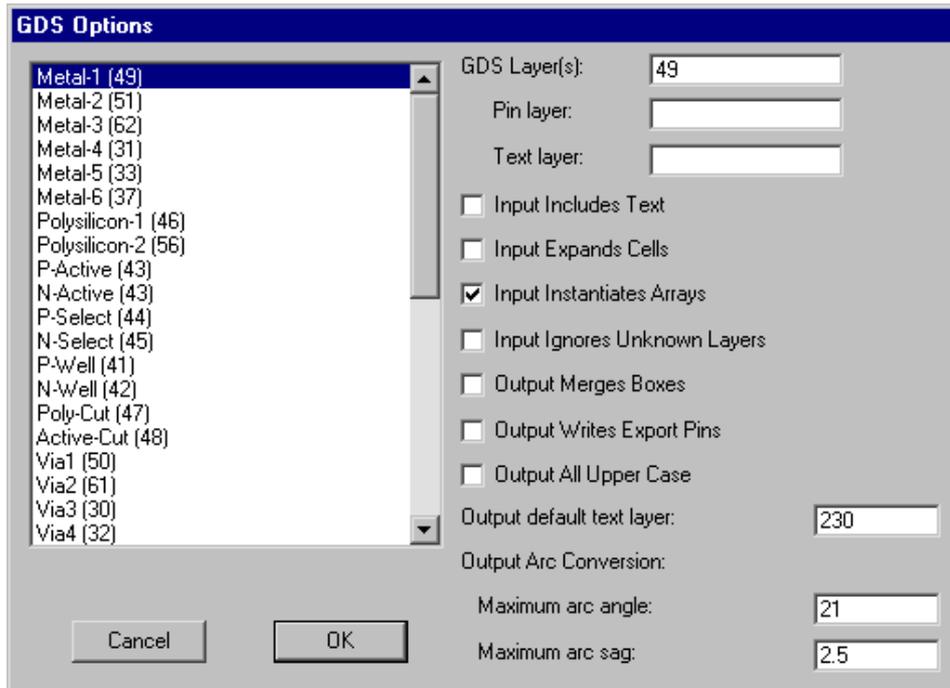
When reading CIF files, the CIF "wire" statements are assumed to have rounded geometry at the ends and corners. If you check the "Input Squares Wires" item, CIF input assumes that wire ends are square and extend by half of their width.

You can request that resolution errors be checked during CIF output. In addition, by selecting "Show Resolution Errors", the errors are can be reviewed, one at a time, after CIF output is done. The "Output resolution" field is the minimum coordinate value that can be safely written in CIF (due to CIF's use of the centimicron as the smallest unit). All geometry of that size or less will be flagged during CIF output. 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.



GDS Control

GDS options are controlled with the **GDS Options...** subcommand of the **IO Options** command of the **File** menu.



In GDS files, there are no names for each layer, just numbers. It is important that Electric know the meaning of each number so that it can properly read and write GDS files. This dialog lets you edit those GDS layer numbers and assign them to different Electric layers.

The list on the left shows all of the Electric layers. By clicking on a layer name, its GDS layer numbers are shown in the top-right and can be edited. You can place multiple GDS layer numbers in the field, separated by commas. When you do this, all of those numbers will be accepted when reading GDS (but only the first will be used when writing GDS). In addition to GDS layer numbers to use for layout, there are also two other types of GDS layer numbers: a *pin* layer (for exports) and a *text* layer (for export names).

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



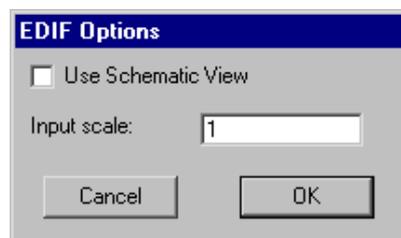
layers are simply ignored.

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 default text layer". This is the layer number to use when writing text. When pins 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.
- "Output Arc Conversion". GDS II format is only able to handle straight lines, not curves. If your design has any curved geometry, the GDS II output will approximate these curves with straight line segments. To control how this approximation is handled, there are two factors: the maximum *angle* and the maximum *sag*. The maximum angle is the number of degrees of curvature between line segments (so a quarter-circle curve, which is 90 degrees, will be broken into 10 lines if the maximum angle is 9 degrees). The maximum sag is the maximum distance between the actual curve and the center of the approximating line segment. No line segment will be allowed to sag more than this distance. These two values can be entered in the "Maximum arc angle" and "Maximum arc sag" areas.

EDIF Control

EDIF options are controlled with the **EDIF Options...** subcommand of the **IO Options** command of the **File** menu.

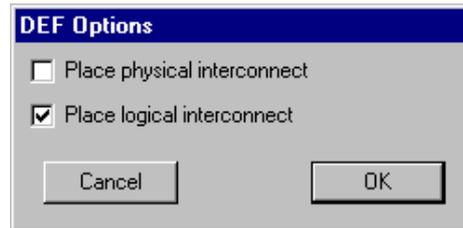


This dialog controls whether EDIF output writes schematic or netlist views (the default is netlist). It also lets you set a scale factor for EDIF input.



DEF Control

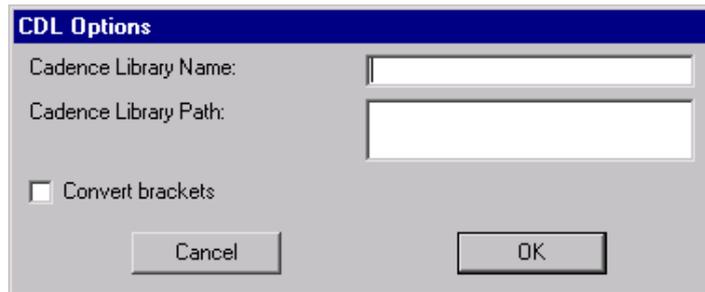
DEF options are controlled with the **DEF Options...** subcommand of the **IO Options** command of the **File** menu.



This dialog controls whether DEF reads physical and/or logical information.

CDL Control

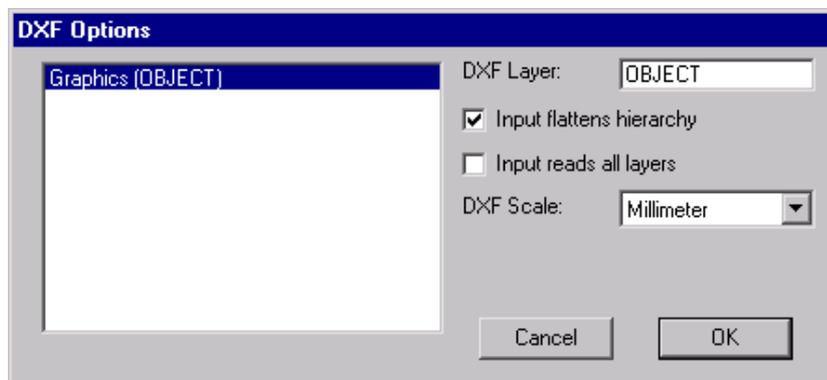
CDL options are controlled with the **CDL Options...** subcommand of the **IO Options** command of the **File** menu.



This dialog control the library name and path information that is written, and it lets you control the conversion of square-bracket characters.

DXF Control

DXF options are controlled with the **DXF Options...** subcommand of the **IO Options** command of the **File** menu.



This dialog controls the conversion between layers in Electric and layers in the DXF file. By clicking on an Electric layer, you can type a new DXF layer into the dialog.



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.

If a layer name in the DXF file is not found in the list that you setup in Electric, 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.

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.

SUE Control

SUE input is controlled with the **SUE Options...** subcommand of the **IO Options** command of the **File** menu.



This dialog controls whether transistors will appear in a standard 3-terminal configuration or in a 4-port configuration with a substrate connection.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 7: DESIGN ENVIRONMENTS



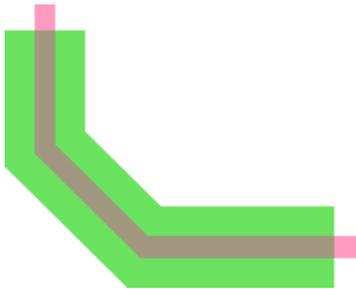
7-4: The MOS Technologies



There are both nMOS and CMOS technologies available in Electric, with many different design rules. Use the **Change Current Technology...** command of the **Technology** menu to select one.

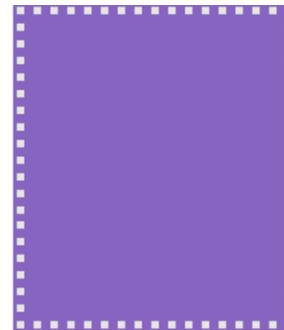
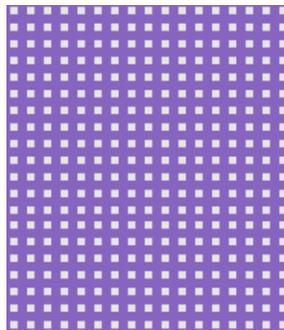
There is one nMOS technology: "nmos" (the specifications used in the Mead and Conway textbook).

There are many more CMOS technologies available. The most basic is "cmos", which uses an idealized set of design-rules from a paper by Griswold. The "mocmos" process has two layers of polysilicon and up to 6 layers of metal with standard, submicron, or deep rules (this is the default technology, and it 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 the **Outline Edit** command of the **Edit** menu (see [Section 6-10](#)).

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 are not 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** subcommand of the **New Special Object** command of the **Edit** menu.

 [Previous](#)  [Table of Contents](#) [Next](#) 



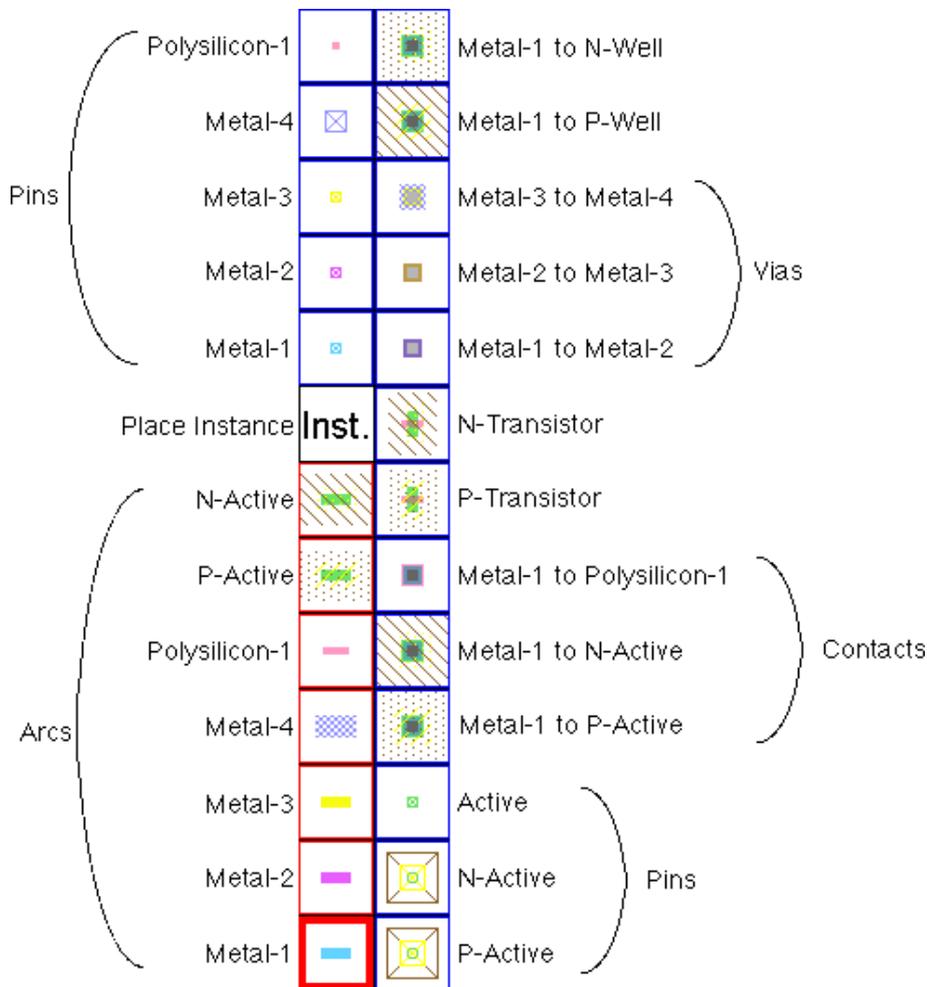
Chapter 7: DESIGN ENVIRONMENTS



7-5: The MOSIS CMOS Technology



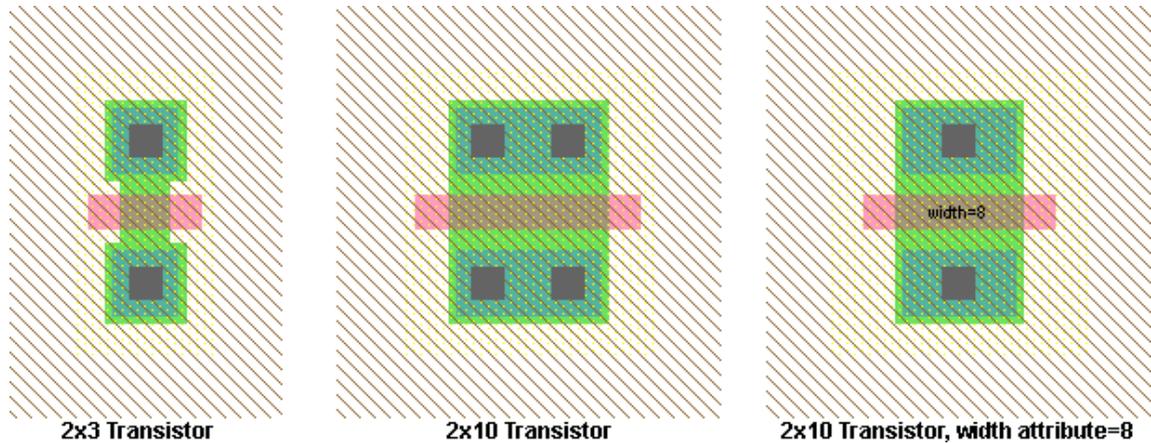
The MOSIS CMOS technology describes a scalable CMOS process that is fabricated by the [MOSIS project](#) of the University of Southern California. To obtain this technology, use the **Change Current Technology...** command of the **Technology** menu and select "mocmossub".



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 Options...** command of the **Technology** menu.



The MOSIS CMOS technology also has two scalable transistor nodes that can be parameterized to have different widths. These transistors are not available by default, check "Show Special transistors" in the **Technology Options...** dialog to see them. (Checking that also provides a vertical PNP transistor that has marginal usefulness.) 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.



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 double-click on a scalable transistor, you get a specialized dialog that allows you to control it. You can choose to have zero or 1 contact, and you can tighten the contact spacing.



Another MOSOS CMOS technology option is to display with "stick figures". This is enabled by using the **Technology Options...** command of the **Technology** menu and checking the "Stick Figures" radio button.

Users of Electric version 6.02 or earlier will have a different MOSIS CMOS technology called "mocmosub". 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, unless you uncheck "Automatically convert to new MOSIS CMOS" in the "MOSIS CMOS Submicron (old)" section of the dialog.



Chapter 7: DESIGN ENVIRONMENTS

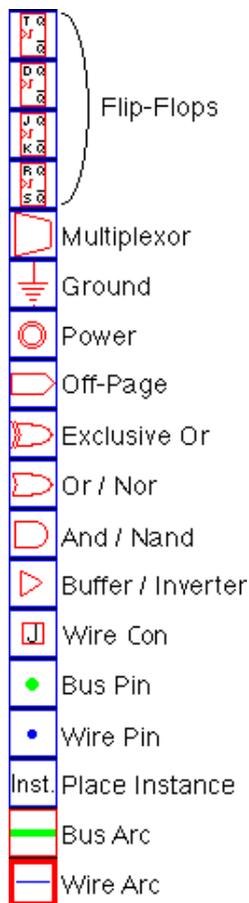


7-6: The Schematic Technology



Digital Schematics

The Schematic technology allows you to design using digital schematic components. To obtain this technology, use the **Change Current Technology...** command of the **Technology** menu and select "schematic, digital".



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](#) 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 Signals** subcommand of the **Network** command of the **Tools** menu 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 **Negated** command of the **Arc** menu), 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 Options...** command of the **Technology** menu and setting the "Negating Bubble Size" value in the "Schematics" section.

The And, Or, and Xor nodes can accept any number of input connections on the left, so they require some care in wiring. 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. The Multiplexor node also has a variable-sized port on the left side.

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.

Analog Schematics

To do analog schematic design, use the **Change Current Technology...** command of the **Technology** menu and select "schematic, analog". The component menu will present a selection of analog schematic nodes. Even without switching to this technology, analog components are always available from the **New Analog Part** and **New SPICE Part** commands of the **Edit** menu.

DMOS Transistor			Black Box
nMOS Transistor			PMOS Transistor
NPN Transistor			PNP Transistor
DMES Transistor			EMES Transistor
NJFET Transistor			PJFET Transistor
4-Port Transistors			3-Port Transistors
Diode			Zener Diode
Capacitor			Electrolytic Capacitor
Resistor			Inductor
Power			Ground
Global			Switch
Off-Page			Wire Con
Spice Part			Place Instance
Wire Pin			Bus Pin
Wire Arc			Bus Arc

The analog nodes can have user-settable values displayed on them. When a node is created, you may be prompted for an appropriate value.

Transistors can be 3-port or 4-port (with bias), and are switched with the "3-Port" and "4-Port" entries.

The Switch node can take an arbitrary number of poles. Simply stretch it along the line of the poles and their number will grow. To do this, use the **Size** command of the **Edit** menu.

The "Spice" entry presents a popup menu of Spice parts (the same as is found in the **New SPICE Part** command). More information about the use of these parts can be found in the [Section 9-4](#).

The "Inst." entry presents a popup menu of all cell instances.

The "Global" node defines a global signal name that spans levels of hierarchy (see [Section 6-9](#)).

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 Options...** subcommand of the **Network** command of the **Tools** menu 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 Options...** command of the **Technology** menu and set the "Use Lambda values from this Technology" field.

Multipage Schematics and Frames

Multipage schematics are provided in Electric by having different views for each page. Thus, you can have cells called "Timer{p1}" and "Timer{p2}" which are pages 1 and 2 of the Timer schematic. To create these cells, use the **Edit Multi-page Schematic View...** command of the **Views** menu.

As a graphical aid to schematic design, frames can be displayed in a cell by using the **Frame Options...** command of the **View** menu.

The "Half-A", "A", "B", "C", "D", and "E" size frames are available in both landscape and portrait mode. You can also choose to display a title box in the lower-right corner.

Frame Options

For cell: inv2iHT{sch}

Frame size: Half-A-Size

Landscape
 Portrait

Title Box

Cancel

OK

Default: Library: redFour

Company Name: Static Free Software

Designer Name: Steven Rubin

Project Name: HDL

In the title box, the cell name and date are shown. You can also add information about the designer, company, and project. This information can be set as a default for all design, and can be overridden on a per-library basis.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 7: DESIGN ENVIRONMENTS



7-7: The Artwork Technology



The Artwork technology is an unusual technology that provides general-purpose sketching facilities. To obtain this technology, use the **Change Current Technology...** command of the **Technology** menu and select "artwork".

Thicker Circle			Spline
Crossed Box			Arrow Head
General Pin		Export	Export
Filled Circle			Circle
Filled Polygon			Closed Polygon
Filled Box			Box
Filled Triangle			Triangle
Opened Thicker Polygon			Opened Dashed Polygon
Opened Polygon			Opened Dotted Polygon
Place Instance	Inst.	Text	Text
Thicker Arc			Dashed Arc
Solid Arc			Dotted Arc

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. Text is created with the **Text (nonlayout)** subcommand of the **New Special Object** command of the **Edit** menu. 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.

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** command of the **Edit** menu to describe them (see [Section 6-10](#)).

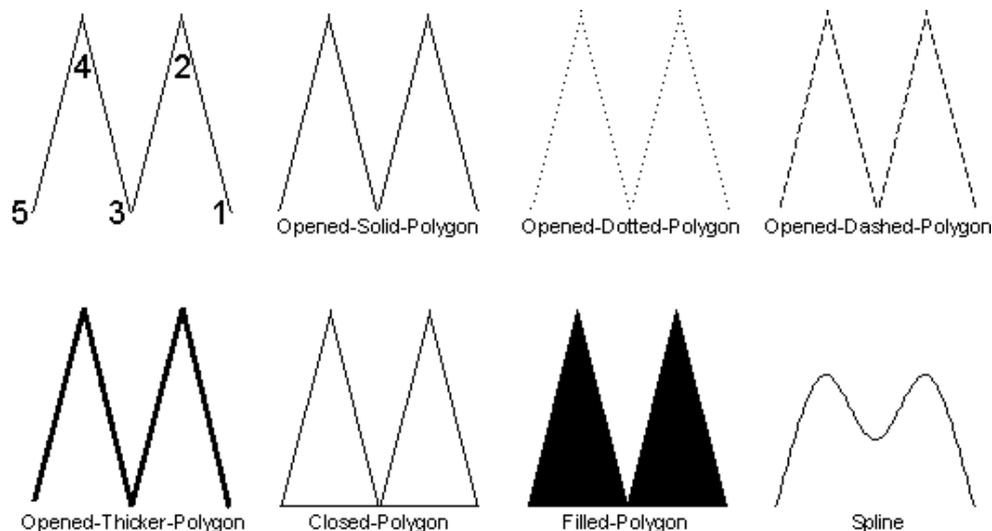
Circles can be outlines (normal or thick) or filled. The default shape is round, but elongation of the node produces an ellipse. In addition, by using the **Get Info** command of the **Info** menu, the outline circles can be reduced to a portion of the circle (from 1 to 360 degrees).

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 Options...** command of the **Technology** menu and set the "Arrows filled" checkbox in the "Artwork" section to control this feature.



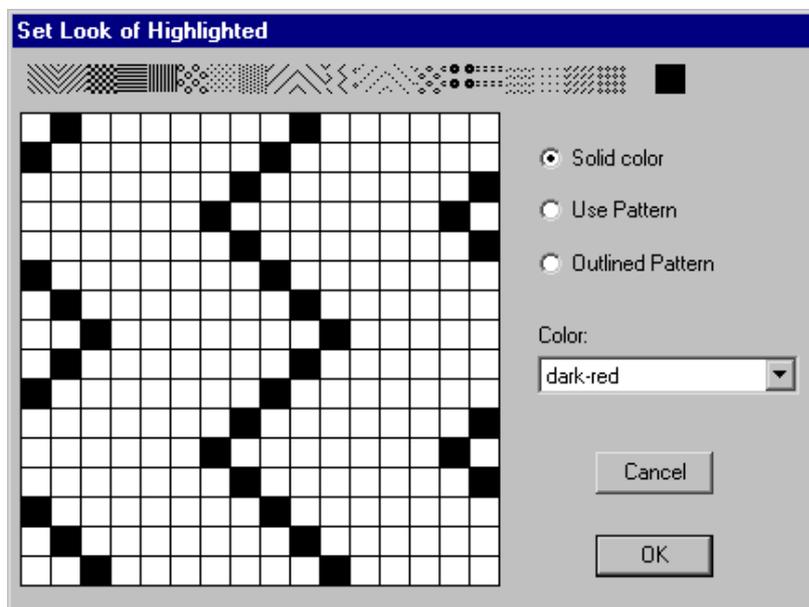
The "Export" entry creates an export for use in icons. After clicking on them entry, you have the choice of selecting "Wire", "Bus", or "Universal" exports (see the **Icon Options...** command of the **View** menu).

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 color of any of its nodes or arcs. Use the **Get Info** command of the **Info** menu to set the color of any node or arc.

You can also use the **Artwork Color...** subcommand of the **Special Function** command the **Edit** menu to change the color or pattern. Predefined patterns are available along the top of the dialog. 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).



[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 7: DESIGN ENVIRONMENTS



7–8: The FPGA Technology



The FPGA technology is a "soft" technology that creates primitives according to an FPGA Architecture file. Once you switch to the FPGA technology, the component menu lists a set of commands for configuring the technology and for programming FPGA circuits.

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 *Architecture* 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.

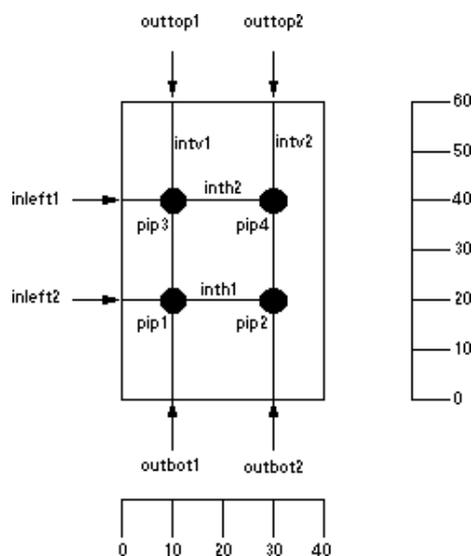
For example, this block has two vertical nets and two horizontal nets. Four pips are placed at the intersections. Six ports are defined (two on the left, two on the top, and two on the bottom). The code is as follows:

```

(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 **blockdefs**, defining a complete hierarchy.

The **attributes** section defines general information about the block.

The **ports** section defines external connections.

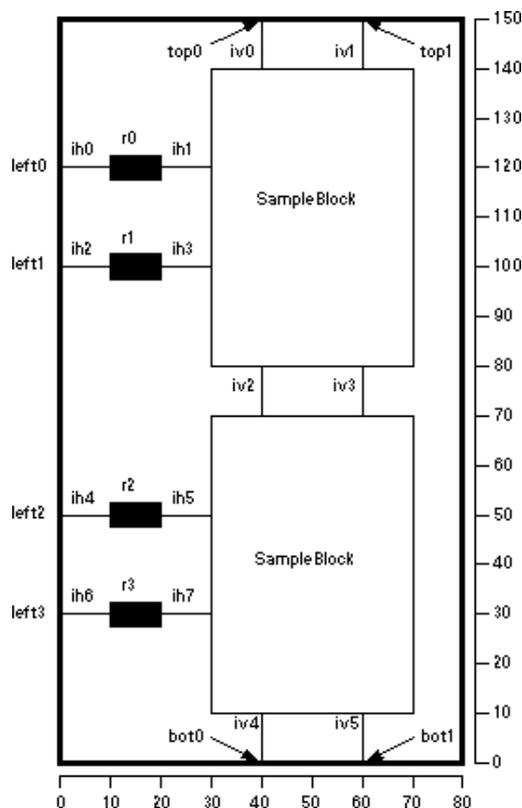
The **components** section defines logic in the block (currently instances of other blocks or repeaters). The **rotation** of an instance is the number of degrees counterclockwise, rotated about the center. The **attributes** section of the instance assigns name/value pairs (this can be used to program 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) )
)

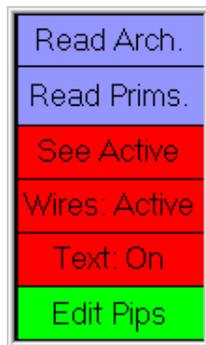
(nets
(net (name iv0)
(segment port top0 component block0 outtop1) )
(net (name iv1)
(segment port top1 component block0 outtop2) )
(net (name iv2)
(segment component block0 outbot1 component block1 outtop1))
(net (name iv3)
(segment component block0 outbot2 component block1 outtop2))
(net (name iv4)
(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, click on the "Read Arch." entry in the component menu. You will be prompted for an architecture file. To read only the primitives from an architecture file, use the "Read Prims." entry.



The display-level can be controlled by clicking on the "Wires:" entry. Its state can be set to "Full" (to see all wires), "Empty" (to show no wires), or "Active" (to show the active wires inside of primitives). Active wires are those connected to PIPs that have been programmed. The "Text:" entry sets the display of text on primitives and can be either "On" or "Off". If you highlight an area of the FPGA chip and click "See Active", then the area will be redisplayed, showing only the active segments.

Once an FPGA has been created, you can program the PIPs by selecting a component and clicking on the "Edit Pips" entry. 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.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 7: DESIGN ENVIRONMENTS



7–9: 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 of the **Edit** menu. Also, the unrouted arc can be selected for subsequent wiring with the **Get Unrouted Wire** subcommand of the **Routing** command of the **Tools** menu.

Special Nodes

There are also special nodes in the Generic technology. 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 the **Text (nonlayout)** subcommand of the **New Special Object** command of the **Edit** menu). This pin can also connect to any arc.

A special primitive, called *Facet–Center*, defines the origin of any cell. It is also available from the **Cell Center** subcommand of the **New Special Object** command of the **Edit** menu. Once the node is placed, instances of the current cell use this node's location, rather than the lower–left corner, as the *grab point* for cursor–based references. 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. Deleting this node restores the lower–left corner as the grab point. 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. This primitive is also available from the **Essential Bounds marker** subcommand of the **New Special Object** command of the **Edit** menu.

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 the *special select* button. Use the **Get Info** command of the **Info** menu, and check "Easy to Select", to allow simple selection of these components.

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 fills–in these probe nodes. It provides a visual display of simulation activity, and works especially well when the VCR controls in the waveform window are used.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 8: CREATING NEW ENVIRONMENTS



8-1: Introduction to Technology Editing



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 **Technology** menu aid in the manipulation of these libraries.

There are four types of cells in a technology library which describe the *layers*, *nodes*, *arcs*, and *miscellaneous-information*. 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 miscellaneous information cell is always called "factors". Any other cell in the library is ignored.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 8: CREATING NEW ENVIRONMENTS



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 and Edit Technology...** command of the **Technology** menu and select a similar technology. Note that 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.

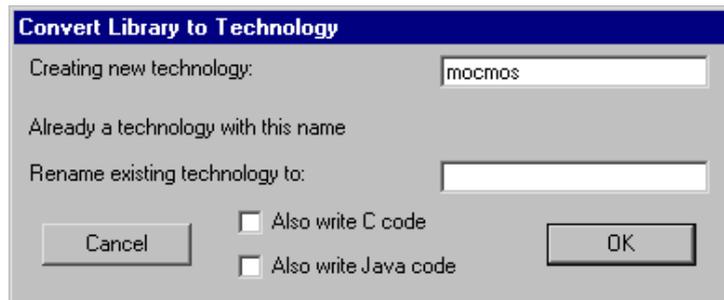
Technology-Editing Mode

Once a technology-library has been created, editing of its cells is done in a special *technology-editing mode* (as a reminder, the cursor changes to a "T" and a yellow border is drawn in the window). If the mode does not appear properly, use the **Cell Options...** command of the **Cells** menu to tell the system whether or not a cell should be edited in the technology editor.

Converting Libraries to Technologies

When in technology-editing mode, the **Convert and Edit Technology...** command changes to the **Convert Library to Technology...** command. This command converts the current library back into a technology.

You are given the opportunity of naming the technology, and can also request that C or 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 (note that technologies can also be renamed with the **Rename Technology...** command).



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. No two technologies can have the same name, so the library name will be adjusted if necessary to form a unique technology name.

Before creating any circuitry with the new technology, it is advisable to create a new library (use the **New Library...** command of the **File** menu) 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).

To delete a technology, use the **Delete Technology...** command of the **Technology** menu. These commands are not undoable.

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, all in one step, with the **Load Technology Library** command of the **Technology** menu.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 8: CREATING NEW ENVIRONMENTS



8-3: Hierarchies of Technology Libraries



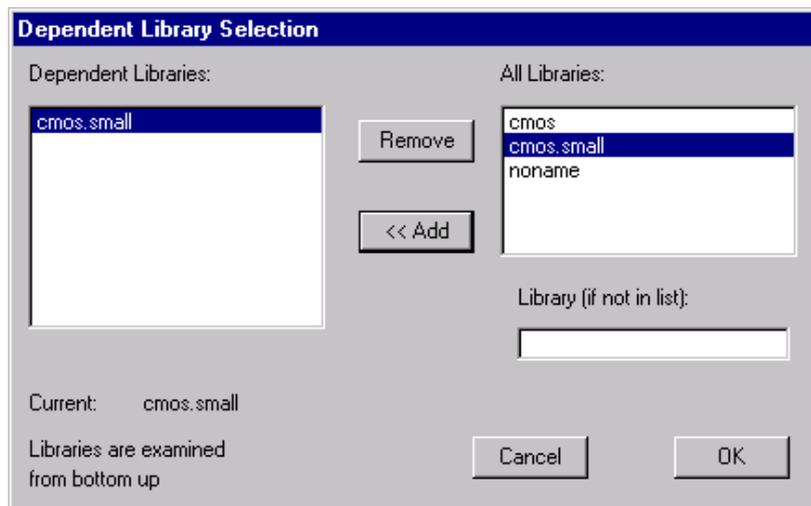
Although a technology is normally described with a single library, it is also possible to string together a sequence of libraries to describe a technology. The sequence is an "inheritance hierarchy", where later libraries in the sequence can override elements found in earlier libraries. For example, 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 and some design rules. That information would then override or augment the base library.

To connect a string of libraries, a list is placed in the bottommost library pointing to the earlier, or "dependent" libraries. In the above example, the "tailoring" library would have the list in it, and the list would point to the "base" library. Note that the list implicitly begins with the current library, and continues in reverse order (that is, after the current library, the last library in the list is considered, then the next-to-last, and so on up to the first library in the list).

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 of the **Technology** menu, which must be issued when editing the bottommost library.

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.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 8: CREATING NEW ENVIRONMENTS



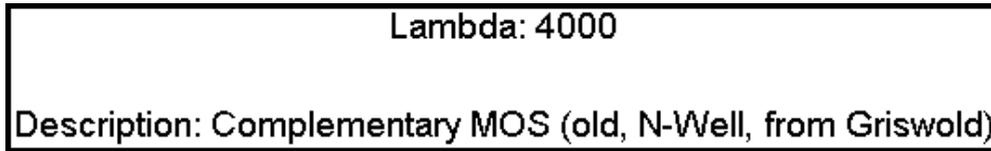
8-4: Miscellaneous Information



The Miscellaneous Information Cell

Each cell in a technology library describes a different aspect of the technology. The *miscellaneous information* cell contains technology-wide information. To see this, use the **Edit Miscellaneous Info** command of the **Technology** menu. Note that this cell is called "factors", so the same effect could be accomplished by selecting it with the **Edit Cell...** command of the **Cells** menu.

The miscellaneous information cell contains two items: the value of lambda (the basic grid unit, measured in internal units), and a full description of the technology. By highlighting any of these items and using the *technology edit* button, the item can be appropriately modified.

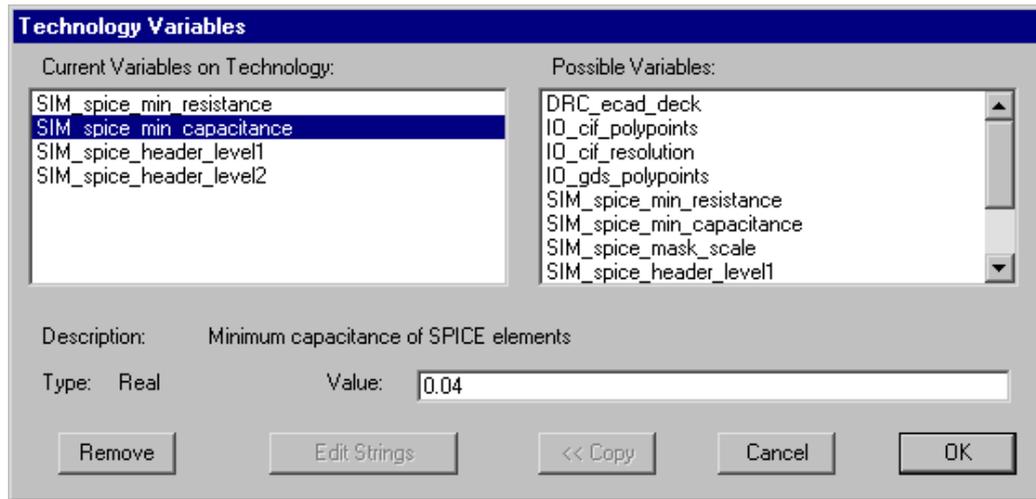


Note that the proper way to set the value of lambda for a technology is by changing this value, and not by using the **Change Units...** command of the **Technology** menu. The **Change Units...** command is only going to scale the examples in the technology library, and will not correctly alter the size of components in the new technology.



Additional Variables

Besides the information shown in this cell, there are many optional *variables* that may be specified with the **Edit Variables...** command of the **Technology** menu.



This dialog shows two lists of variables that can be attached to the technology. On the left is a list of variables that are currently attached, and on the right is a list of all known variables that can be attached to the technology. To add a new variable to the technology, select it from the list on the right and click the "<< Copy" button. To remove a variable from the technology, select it in the current list and click the "Remove" button. When a variable in either list is selected, its description and type are shown. When the variable is of type "Integer", "Real", or "String", you may change its value in the "Value" field. When you have selected a variable of type "Strings" (note the "s") then it must be edited in a separate window with the "Edit Strings" button. The "Edit Strings" button first exits this dialog and then opens a text edit window for manipulating the variable. See [Section 4–10](#) for more on text editing.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 8: CREATING NEW ENVIRONMENTS



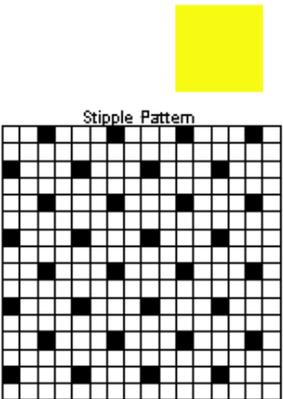
8-5: The Layer Cells



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, use the **Edit Layer...** command of the **Technology** menu, and to create a new layer use the **New Layer...** subcommand of the **New Primitive** command. Once a layer is being edited, the next one in sequence can be edited with the **Edit Next Primitive** command. The current layer can be deleted with **Delete this Primitive**. Also, you can rename a layer with the **Edit Cell...** command of the **Cells** menu, 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 by using the **Reorder Layers** subcommand of the **Reorder Primitives** command.

There are many pieces of information in a layer, most of which can be updated by highlighting them and using the *technology edit* button. 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.



Color: transparent-5
Style: solid
CIF Layer: CMT
DXF Layer(s):
GDS-II Layer: 62
Function: metal-3
Layer letters: r
SPICE Resistance: 0.06
SPICE Capacitance: 0.04
SPICE Edge Capacitance: 0
3D Height: 21
3D Thickness: 0
Print colors: 255,255,51, 72,on

The stipple pattern can be changed simply by selecting any grid squares and then using the *technology edit* button. You can also do operations on the entire stipple pattern by clicking on the text "Stipple Pattern Operations" and using the *technology edit* button. These operations include clearing the pattern, inverting the pattern, copying, and pasting the pattern (between layers).





The color of the layer can be changed by highlighting the "Color" entry on the right and using the *technology edit* button. A menu then proposes five transparent and many named (opaque) colors.

Transparent colors have a unique appearance when they overlap each other, as defined in the technology's color map. The named colors are opaque, so they obscure anything under them when drawn. In general, the five most commonly used layers should be assigned to the five transparent colors, and the remaining layers should have opaque color. See [Section 4–6](#) for more information on color.

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.

The "Print colors" at the bottom are used when producing PostScript that is "Color Merged" (see the **Print Options...** command of the **File** menu). The first three numbers are the red, green, and blue. The next number is an opacity, from 100 (fully opaque) to 0 (transparent). The last factor is a "foreground" flag which, when "on" indicates that the non-opaque colors can be combined with others.

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 "DXF Layer" entry is the string to use for DXF I/O. The "GDS-II layer" is a single number (–1 when there is no layer).



The "Function" entry allows a general-purpose description to be attached to the layer. A function can be only 1 of the values from this table. However, the last 12 entries are additional attributes that can combine with the layer function to further describe it:

- The functions "p-type," "n-type," "depletion," "enhancement," "light," and "heavy" describe layer types that are process-specific.
- The function "pseudo" indicates that this layer is a pseudo-layer, used for pin construction.
- The function "nonelectrical" indicates that this layer is decorative and not part of a real circuit.
- The functions "connects-metal," "connects-poly," and "connects-diff" indicate that this contact layer joins the specified real layers.
- The function "inside-transistor" indicates that the polysilicon is not field-poly, but is part of a transistor.

For example, you can highlight the function entry and use the *technology edit* button 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."

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 with "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.

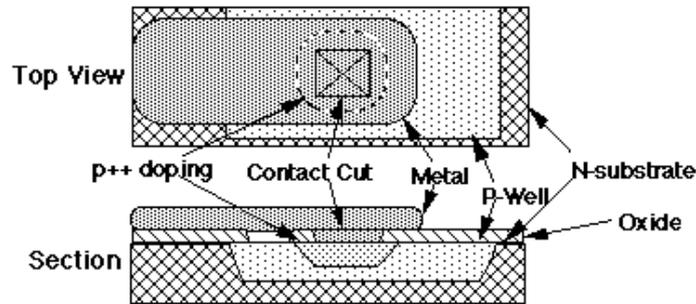
new function for this layer
unknown
metal-1
.....
metal-12
poly-1
.....
poly-3
gate
diffusion
implant
contact-1
.....
contact-12
plug
overglass
resistor
capacitor
transistor
emitter
base
collector
substrate
well
guard
isolation
bus
art
control
p-type
n-type
depletion
enhancement
light
heavy
pseudo
nonelectrical
connects-metal
connects-poly
connects-diff
inside-transistor

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; and (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".



The "Layer letters" field is one letter (or at most two) that uniquely identify this layer. You must not place the same letter in this field on two different layers. These letters are used as abbreviations for the layers in some internal commands, but have no meaning to the average user. Nevertheless, it is best to keep track of which letter has been used with each layer to prevent duplication.

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. If a layer has nonzero edge capacitance, it must also have nonzero capacitance, because the extractor checks the capacitance value to decide whether to include the layer in parasitic computations. This is true even if the layer is used in an arc that contains device diffusion; the extractor will correctly cancel out the capacitance, and include the edge capacitance in the extraction process.

The "3D Height" and "3D Thickness" are used when viewing a chip in 3-dimensions. The height and thickness are arbitrary values (usually a small integer). They indicate 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. Use the **3D Display** commands of the **Windows** menu to see a 3-dimensional view of the circuit.

[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 8: CREATING NEW ENVIRONMENTS



8-6: Special Layer Information



Once the layers have been defined, the color map can be built to describe the transparent colors and all of their combinations. Use the **Edit Colors...** command of the **Technology** menu to modify all 32 combinations of the five transparent layers. A color mixing palette is presented in which any of the five transparent layers can be viewed in conjunction with the other four. It is necessary that exactly five layers be given transparent status (ten if there are associated pseudo-layers). This editing facility is the same one that is found in the **Edit Colors...** command which modifies the current display (see [Section 4-6](#)).

Another piece of information that can be determined, once the layers have been defined, is the design rules. Use the **Edit Design Rules...** command of the **Technology** menu to see them. This command displays a dialog similar to the one that modifies rules of existing technologies (see [Section 9-2](#) for more on design-rules)

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 8: CREATING NEW ENVIRONMENTS



8-7: 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, use the **Edit Primitive Arc...** command of the **Technology** menu, and to create a new arc use the **New Primitive Arc...** subcommand of the **New Primitive** command. Once an arc is being edited, the next one in sequence can be edited with the **Edit Next Primitive** command. The current arc can be deleted with **Delete this Primitive**. Also, you can rename an arc with the **Edit Cell...** command of the **Cells** menu, but remember to use the name "arc-" in front (i.e. the old name is "arc-polysilicon" and the new name is "arc-gate"). Finally, you can rearrange the order in which the arcs will be listed by using the **Reorder Primitive Arcs** subcommand of the **Reorder Primitives** command.

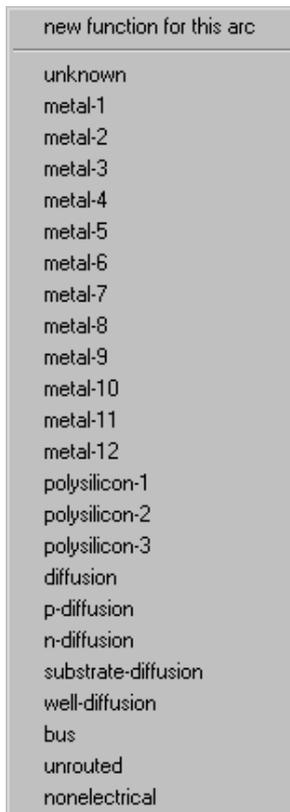


Editing Special Arc Information

Arc cells show a sample arc on the left and a few pieces of textual information on the right. The textual information can be updated by highlighting them and using the *technology edit* button.



Function: metal-1
Fixed-angle: Yes
Wipes pins: Yes
Extend arcs: Yes
Angle increment: 90



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.

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).

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).



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.

PORT	Ports (nodes only)
HIGH	Highlight Box
	Filled Box
	Outlined Box
	Crossed Box
	Filled Polygon
	Outlined Polygon
	Opened Polygon
	Opened Dotted Polygon
	Opened Dashed Polygon
	Opened Thicker Polygon
	Opened Circle
	Filled Circle
HALF	Opened Half-Circle
ARC	Opened Arc of Circle
TEXT	Text

Use the entries from the component menu on the left 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 highlighting it and using the *technology edit* button. A menu is then presented with possible layers (ignore the last entry, "SET-MINIMUM-SIZE", which is used only for nodes).

new layer for this patch
Metal
Polysilicon
Diffusion
PP
Contact_Cut
Ohmic_Cut
P_Well
Overglass
Transistor
Pseudo_Metal
Pseudo_Polysilicon
Pseudo_Diffusion
Pseudo_PP
Pseudo_P_Well
SET-MINIMUM-SIZE

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, which temporarily labels each piece of geometry in the arc cell.

[< Previous](#)

 [↑ Table of Contents](#)

 [Next >](#)



Chapter 8: CREATING NEW ENVIRONMENTS



8-8: 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, use the **Edit Primitive Node...** command of the **Technology** menu, and to create a new node use the **New Primitive Node...** subcommand of the **New Primitive** command. Once a node is being edited, the next one in sequence can be edited with the **Edit Next Primitive** command. The current node can be deleted with **Delete this Primitive**. Also, you can rename a node with the **Edit Cell...** command of the **Cells** menu, but remember to use the name "node-" in front (i.e. the old name is "node-metal1-metal2-contact" and the new name is "node-via"). Finally, you can rearrange the order in which the nodes will be listed by using the **Reorder Primitive Nodes** subcommand of the **Reorder Primitives** command.

Editing Special Node Information

The node cell contains four pictures of the node on the left and textual information on the right. You can update the textual information entries by highlighting them and using the *technology edit* button.

Function: contact
Serpentine transistor: no
Square node: no
Invisible with 1 or 2 arcs: no
Lockable: no
Multicut separation: 3

The "Function" entry describes the node's function, which is a different set than the arc and layer functions.



The menu of possible node functions is shown here.

The "Serpentine transistor" entry indicates that this is a MOS transistor and it can take arbitrary outline information to describe its geometry.

new function for this node	
unknown	electrolytic-capacitor
pin	diode
contact	zener-diode
pure-layer-node	inductor
connection	meter
nMOS-transistor	base
DMOS-transistor	emitter
pMOS-transistor	collector
NPN-transistor	buffer
PNP-transistor	AND-gate
n-type-JFET-transistor	OR-gate
p-type-JFET-transistor	XOR-gate
depletion-mesfet	flip-flop
enhancement-mesfet	multiplexor
prototype-defined-transistor	power
transistor	ground
4-port-nMOS-transistor	source
4-port-DMOS-transistor	substrate
4-port-pMOS-transistor	well
4-port-NPN-transistor	artwork
4-port-PNP-transistor	array
4-port-n-type-JFET-transistor	align
4-port-p-type-JFET-transistor	ccvs
4-port-depletion-mesfet	cccs
4-port-enhancement-mesfet	vcvs
4-port-transistor	vccs
resistor	transmission-line
capacitor	

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.

The "Multicut separation" is the distance between multiple contact cuts when the node grows large. Although this distance can be automatically determined from the different examples, it can also be give explicitly here, and this allows you to draw only one example.

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 on the left, creating the geometry, and then using the *technology edit* button to assign a layer to the geometry. If any polygonal geometry is used (for



example, the Filled polygon entry, sixth from the top), they require outline information to be assigned with the **Outline Edit** subcommand of the **Special Function** command of the **Edit** menu. If the Opened circle arc entry is selected (second from the bottom), you can specify the number of degrees of the circle with the **Get Info** command of the **Info** menu.

Each example must also contain a highlight layer to indicate the correct highlighting on the display. Select the "HIGH" entry in the menu on the left 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). Use the *technology edit* button to set this (see the sample menu on the right). Each possible arc listed can have its connectivity set by typing "y" or "n" when pointing to its name in the menu.

options for this port	
Metal_1	yes
Metal_2	no
Polysilicon	no
S_Active	no
D_Active	yes
Active	no
PORT-ANGLE	0
PORT-ANGLE-RANGE	180

The range consists of two numbers: a main angle (in degrees counterclockwise from 3 O'clock) and a range about that angle. 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.

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. Use the *selection* and *toggle select* buttons on the two ports, then use the *technology edit* button to 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.

Although the "grab point" is usually defined to be the center of the highlight area (unless the user unchecks "Center-based primitives" in the **Selection Options** dialog), this can be overridden by placing a Cell-Center node (named "Facet-Center" for historical reasons) at the appropriate location in the main example. Placing this mark on the left side will cause instances of the node to be placed by their left side rather than the center of the highlight area. To get this node, use the **Cell Center** subcommand of the **New Special Object** command of the **Edit** menu.

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 (with the *technology edit* button) 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 size and spacing 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 field on the right. 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.

[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 8: CREATING NEW ENVIRONMENTS



8–9: 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 general information on the technology (the Miscellaneous Info cell, design-rules, color maps, and variables). 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 general information

Adding information has no effect on the existing circuitry. All subsequent circuit design may make use of the new technology elements.

When adding layers, it is possible that existing layer tables will no longer be valid. Layer tables are those tables that are associated with layers, for example the CIF or GDS layer associations. Because these layer tables are part of the saved options, they override the specifications in the technologies. Therefore, individual users who have customized these tables may have incorrect information when the technology is modified.

If layers are added to the end, then the tables will simply have empty entries for the new layers. If, however, layers are added in the middle, the entries in the layer tables will now be associated with the wrong layer. You will have to edit the layer tables and fix the entries.

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.

Layer deletion can cause the same problem that layer addition presents: inconsistent layer tables. When deleting layers, the entries in the layer tables will now be associated incorrectly. You will have to edit the layer tables and fix the entries.



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 components.

Deleting general 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.

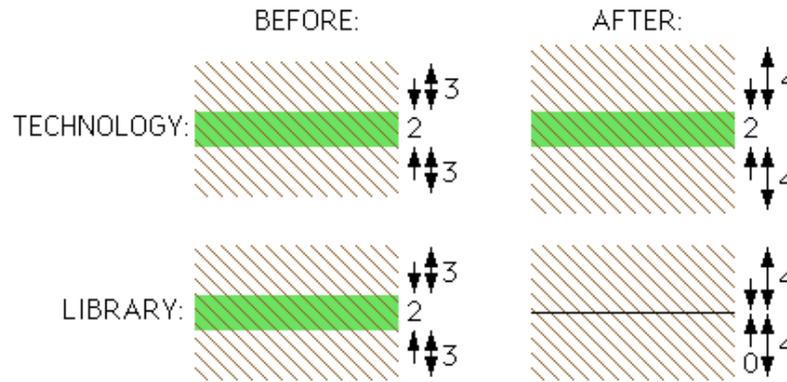
Many of the pieces of information on the right side of the node and arc cells apply to newly created circuitry only, and do NOT affect existing components already in libraries. The arc factors "Fixed-angle", "Wipes pins", "Extend arcs", and "Angle increment" have no effect on existing libraries. The 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, passes 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 4–wide strips where they used to be 2–wide.

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 Layout View...** command of the **View** menu 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 general 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 headers will be seen when a SPICE deck is next generated. A change to the design rules will be used even on the old libraries.

There is one exceptional piece of information that does NOT change in existing libraries: the value of Lambda in the Miscellaneous Information cell. A change to this value does cause the technology to change, and therefore all subsequently created libraries will have this new value. However, if you read any old libraries, the former value of lambda will be used for that library. Also, because the value of lambda in the technology is stored in the libraries, any new circuitry created in this library will also have the old value of lambda. Only by switching libraries to one with the current value of lambda (or changing the value of lambda with the **Change Units...** command of the **Technology** menu) can the new value be established.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 8: CREATING NEW ENVIRONMENTS



8–10: Examples of Use

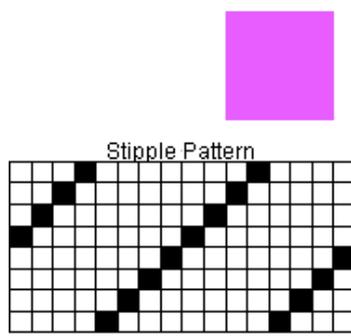


To fully understand technology editing, some examples are appropriate. Two examples will be given: a simple one that modifies the appearance of a pattern, and a more complex example in which a new primitive node is created. Both examples are based on the MOSIS CMOS technology, so they presume that the **Convert and Edit Technology...** command of the **Technology** menu has been issued and the "mocmossb" entry was selected.

Example: Modifying a Layer's Look

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 **Layer Display Options...** command of the **Windows** menu, but it illustrates the basic steps of making a change. Once in technology-editing mode, issue the **Edit Layer...** command of the **Technology** menu and select "metal-2". The display will show the layer with all of its associated information.



Color: overlappable-4
~~Style: solid~~
CIF Layer: CMS
DXF Layer(s):
GDS-II Layer: 51
Function: metal-2
Layer letters: h
SPICE Resistance: 0.06
SPICE Capacitance: 0.04
SPICE Edge Capacitance: 0
3D Height: 19
3D Thickness: 0

new style of this layer
solid
patterned
patterned/outlined

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, select the text and use the *technology edit* button. Note that

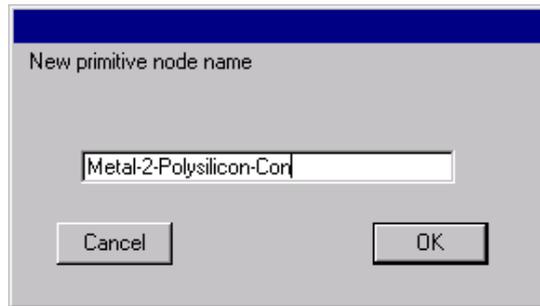


each piece of text has a box around it: you can click anywhere in the box to choose the text (which will be highlighted with an "X" through it). When you press the *technology edit* button, a menu appears that provides three choices: the desired choice is "patterned". The technology is now modified and 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 **New Primitive Node...** subcommand of the **New Primitive** command of the **Technology** menu and name the node appropriately.



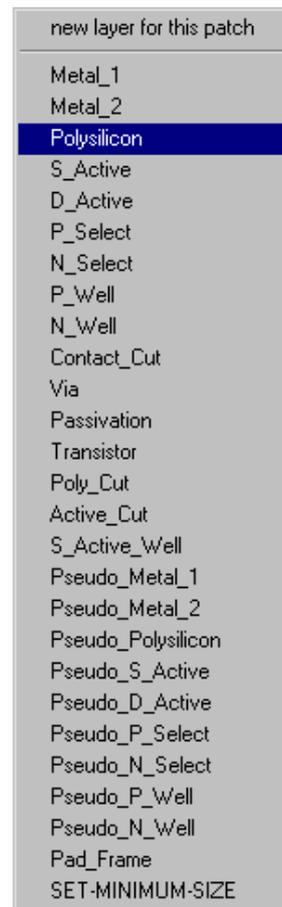
At this point, the display will show only the textual information about the node (because the graphical information is yet to be supplied). The textual information consists of four factors that now fill the screen.

Function: pin
Serpentine transistor: no
Square node: no
Invisible with 1 or 2 arcs: no
Lockable: no
Multicut separation: 0

You should begin by changing the "Function" factor to "contact" (select it, use the *technology edit* button, and choose 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 menu on the left (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, use the *technology edit* button (while the geometry is selected). Then choose "polysilicon". The black box will change appearance to that of a polysilicon layer. You can move and stretch this box appropriately.



In this example, assume that a contact between polysilicon and metal-2 has three layers: polysilicon, 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 menu on the left. 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).

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, use the *technology edit* button on the port.

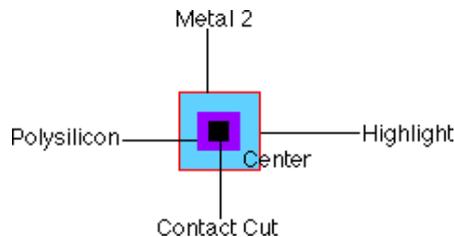
options for this port	
Metal_1	no
Metal_2	no
Polysilicon	yes →
S_Active	no
D_Active	no
Active	no
PORT-ANGLE	0
PORT-ANGLE-RANGE	180

The resulting menu lists all of the arcs and indicates possible connectivity with a "yes" or a "no". To allow arcs to connect, simply move the cursor to that arc and type "y". This can be done repeatedly before clicking to dismiss the menu. 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.

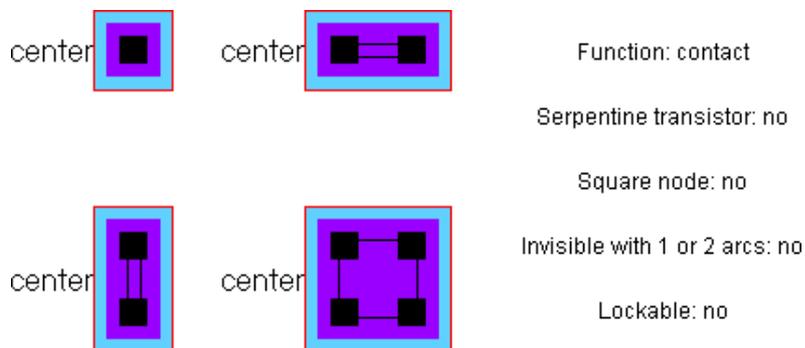
When all of the geometry, highlighting, and ports have been placed, you can double-check your work with the **Identify Primitive Layers** command of the **Technology** menu, which will display this information (note



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 of the **Edit** menu. 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:



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 can use the **Load Technology Library** command of the **Technology** menu to restore your custom technology. Note that this must be done **BEFORE** reading any libraries that make use of the custom technology.



Chapter 9: TOOLS



9-1: Introduction to Tools

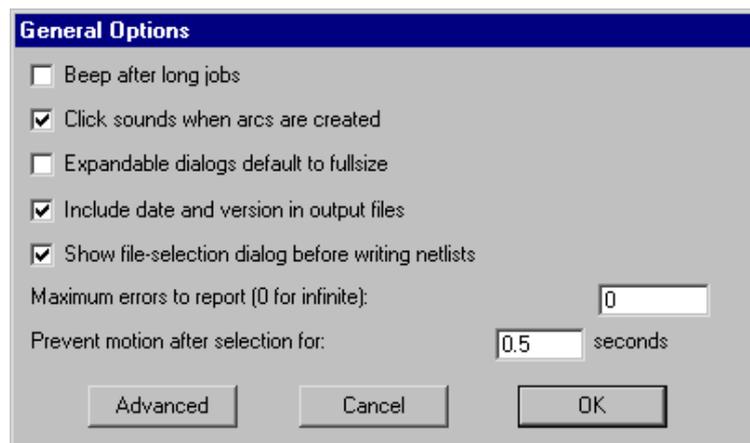


There are many different tools available in Electric for doing both synthesis and analysis of circuitry. Synthesis tools include routers, compactors, circuit generators, and so on. Analysis tools include design-rule checkers, network comparison, and many simulators. To see a list of tools, including which ones are active, use the **List Tools** command of the **Tools** menu. This chapter covers many of the tools available in Electric. [Chapter 10](#) goes into greater detail on the simulation tool.

Overall control of tools is available with the **General Options...** subcommand of the **User Interface** command of the **Info** menu.

By clicking "Beep after long jobs", you can request that the system make a sound after long jobs have finished (those that take more than 1 hour).

When drawing arcs, Electric makes "clicking" sounds for each arc drawn. This can be disabled by unclicking "Click sounds when arcs are created".



Expandable dialogs (currently only the node **Get Info** dialog) start off small. By clicking "Expandable dialogs default to fullsize", these dialogs will start off in their larger size.

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 unclicking "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.

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.

Finally, you can set the delay that is built-in to the click-and-drag action. To prevent mouse-jitter from moving objects when selecting them, the system waits a half second after a click before accepting movements. This delay can be altered here.



The "Advanced" button brings up a dialog of special commands that are not for general usage. If you do click this button, please use the "Cancel" button to terminate the dialog.

 [Previous](#)  [Table of Contents](#) [Next](#) 



Chapter 9: TOOLS



9-2: Design-Rule Checking



The design-rule checker is a collection of tools for checking the spacing of a circuit. By default, there is an *incremental* design-rule checker that watches over the editing session and displays error messages when the geometry is incorrect. A *hierarchical* checker examines the layout hierarchically. In addition to that, it is possible to prepare input decks for Dracula, an external design-rule checker.

Three 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.

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.

Incremental DRC

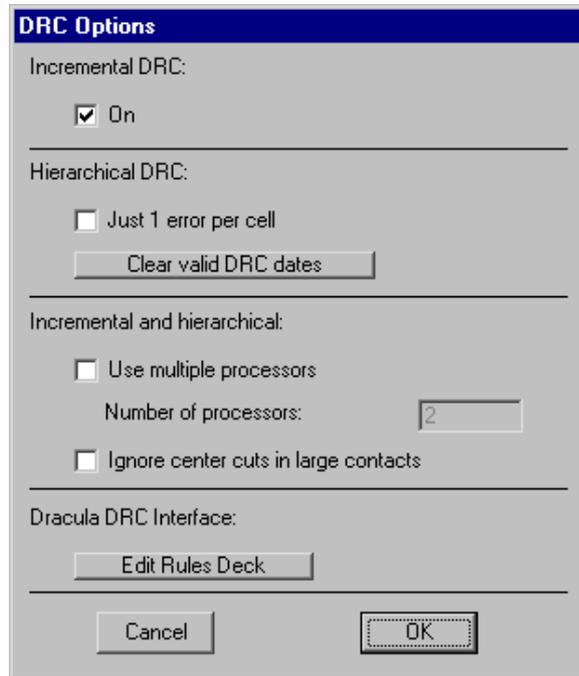
The incremental design-rule checker is always running, examining your work, and issuing error messages when an error is detected. The user should be warned that the incremental design-rule checker does not examine hierarchy. This means that if a cell instance is used in a circuit, the incremental design-rule checker does not examine its contents to see how that interacts with other layout surrounding the instance. To check the complete hierarchy, use the **Check Hierarchically** subcommand.

To control the DRC, use the **DRC Options...** subcommand of the **DRC** command of the **Tools** menu.



By default, the incremental design-rule checker is on. To turn it off, uncheck the "On" checkbox in the "Incremental DRC" section. While the tool is off, Electric keeps track of all cells that change. When the tool is turned back on it rechecks all of those changed cells. Thus, the incremental design-rule checker can be made into a "batch" tool by keeping it off until circuit layout is complete.

MOS contact nodes automatically increase the number of cuts when they grow larger (see Section 7-4). 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.



After errors have been reported, you can review them by typing ">" and "<" to step to the next and previous error that was found. If you wish to recheck an entire cell, use the **Check this Level Only** subcommand.

Hierarchical DRC

The hierarchical design-rule checker uses the same rules and techniques as the incremental checker, but it is able to check across levels of hierarchy. To run it, use the **Check Hierarchically** subcommand of the **DRC** command of the **Tools** menu. To check only a selected subset of the current cell, use **Check Selection Area Hierarchically**.

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 using the **DRC Exclusion** subcommand of the **New Special Object** command of the **Edit** menu. Any errors that fall inside of this node's area are ignored.

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 3-7).

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 Options...** dialog. To see which cells have passed Hierarchical DRC, use the **General Cell Lists...** command of the **Cells** menu (a "D" is shown in on the right for cells that are DRC current).

Another way to speed up Hierarchical DRC is to check the "Just 1 error per cell" entry in the **DRC Options...** dialog. This tells the system to stop checking a cell after the first error has been found. By using



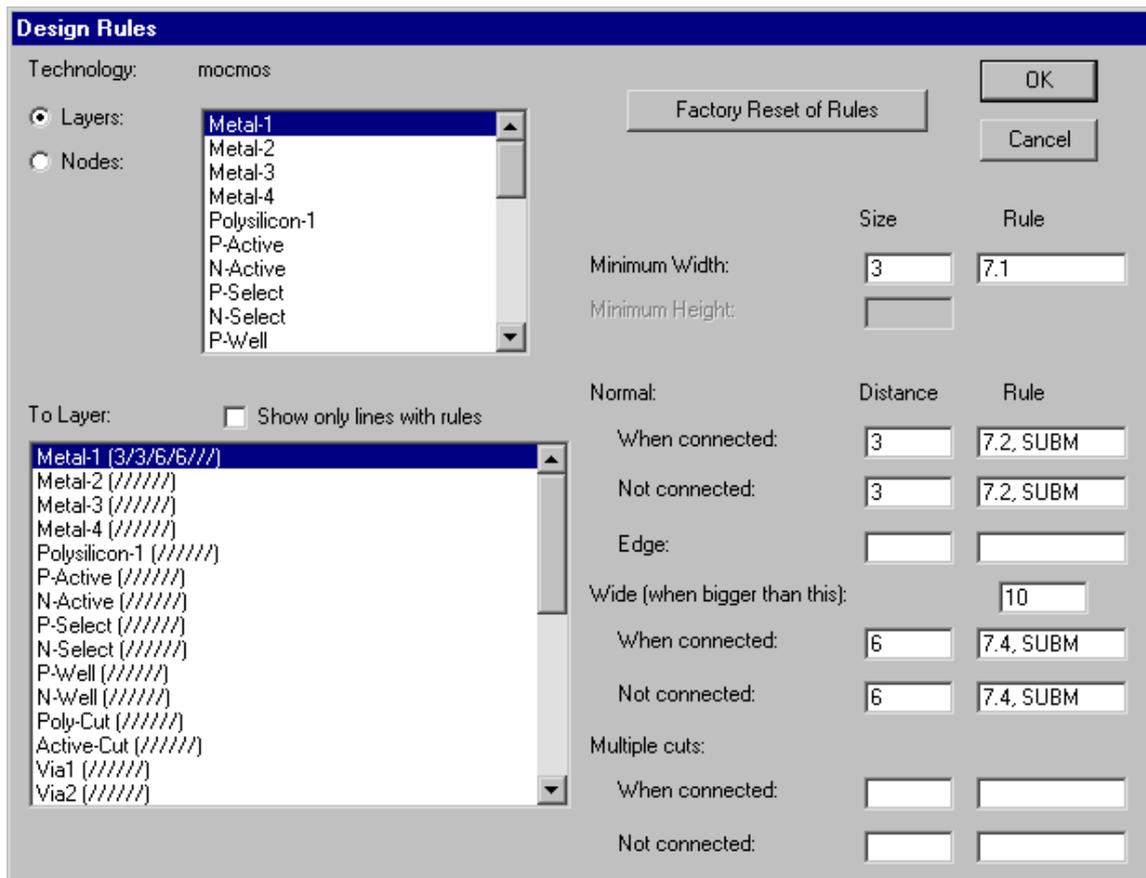
this option, you can more quickly determine *which* cells in the design are correct, without knowing exactly where the errors lie. Then, you can go to the cells with errors and do a more complete check.

If you are fortunate enough to have a computer with more than 1 processor, the DRC can take advantage of this when doing hierarchical checking. Check "Use multiple processors" in the **DRC Options...** dialog and tell it how many processors to use. The value of this field sets the number of parallel threads that will be spawned to do the DRC, so it can be more than the number of processors.

DRC Rules

The **DRC Rules...** dialog allows you to examine and modify the spacing limits for the current technology. You can select "Layers" or "Nodes". When "Nodes" are selected, you may set the minimum size of each node in the current technology.

When "Layers" are selected, you may set the minimum size of each layer as well as inter-layer spacing (between that and the "To Layer"). Use the "Show only lines with rules" to restrict the displayed rules to those with valid values. Each spacing rule comes in two flavors: connected and unconnected. The connected rules apply to two different layers that are electrically connected; the unconnected rules apply to unconnected layers. A special Edge rule applies only to unconnected layers and ignores overlap when considering spacing distance. The connected and unconnected rules come in three styles: normal, wide, and multiple cut. The Wide rules apply when either layer is wider than a specified amount. The Multiple cut rules apply when either layer is part of a multi-cut contact. In addition to specifying a spacing distance, you can give a description of the rule that will be reported by the design-rule checker. The "Factory Reset of Rules" button restores all rules to the original set built into Electric.



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.

Dracula DRC

Another design-rule checking facility that is available in Electric is an interface with the Dracula design-rule checker. This interface requires a circuit description and a set of design rules. Electric knows the design-rules (currently only for the MOSIS CMOS technology) and is able to generate the proper circuit description (a CIF file). To generate these files, use the **Write Dracula Deck** subcommand.

To see the set of Dracula design rules for the current technology, use the "Edit Dracula Deck" button of the **DRC Options...** subcommand. This will display the rules in an edit window. The rules must contain the lines: "PRIMARY =" and "INDISK =" so that the deck generator can substitute the proper file names.

Note that only the "mocmos" technology has valid design rules, so this command will present an empty window when run in other technologies. However, you can create your own design-rules for any technology. To do this, follow these steps:

- Use the **Change Current Technology...** command from the **Technology** menu to switch to the "mocmos" technology.
- Edit the rules using this command.
- Select everything and use the **Copy** command of **Edit** menu to copy them.
- Close the text editing window.
- Use the **Change Current Technology...** command from the **Technology** menu to switch to the desired technology.
- Edit the rules with this command, which displays a blank editing window.
- Use the **Paste** command of the **Edit** menu to recover the MOSIS CMOS rules.

You can now edit these rules, and they will be saved with your options.

To help guide the Dracula design-rule checker, a "cloaking" layer can be placed over areas that are not to be examined. This cloaking layer is created by using the **DRC Exclusion** subcommand of the **New Special Object** command of the **Edit** menu. The node that is placed produces a layer called "DRC" in the Dracula file, which causes the circuitry underneath to be ignored.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 9: TOOLS



9-3: Electrical-Rule Checking



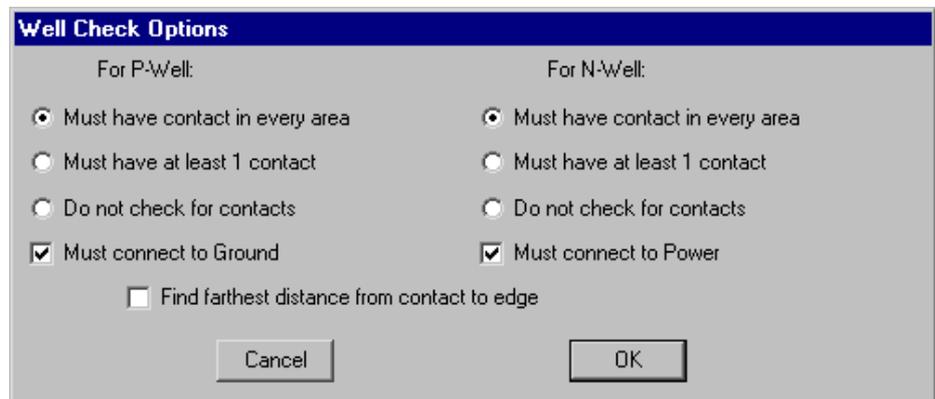
The electrical-rule checker (ERC) is a collection of tools for checking a circuit's behavior without simulation. Currently, two checks are available: well/substrate checking and antenna rules checking.

Well and Substrate Checking

To check the well and substrate layers, use the **Analyze Wells** subcommand of the **Electrical Rules** command of the **Tools** menu. This does a more thorough job of checking the layers than the design-rule checker. This may take some time, and if it takes too long, you can abort it with the interrupt key (see [Section 1-9](#)).

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 3-7](#)).

The Well Checker makes sure that there are well contacts in every area of well. You can relax this restriction with the **Well Check Options...** command. This dialog allows you to request that there be only 1 well contact, anywhere on the chip. You can also instruct the checker to ignore contacts entirely.



The Well Checker also checks spacing rules between well areas. Although this is generally the domain of the Design Rule Checker, it is not done there, so it is done here.

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 Options...** 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.

Finally, the Well Checker reports the maximum distance from a well contact to any point on the well. This is



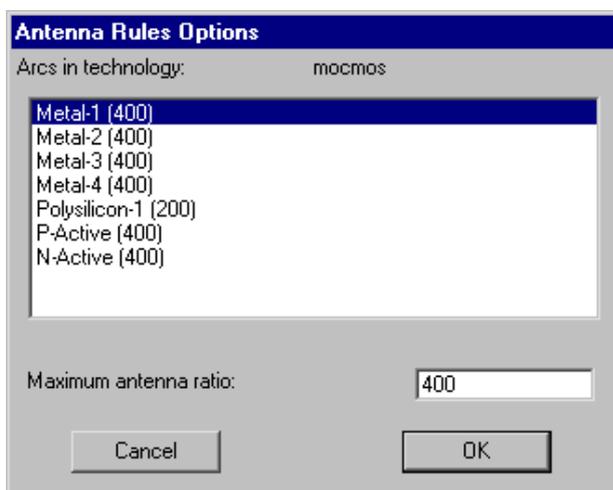
useful when making sure that there are sufficient contacts for each area.

Antenna Rule Checking

Antenna rules are required by some IC manufacturers to ensure that the transistors of the chip are not destroyed during fabrication. During fabrication, the wafer is bombarded with ions during the polysilicon and metal layer creation process. 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–Rules Check** subcommand of the **Electrical Rules** command of the **Tools** menu. 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 3–7](#)).

To modify the required ratio of poly or metal to transistor area, use the **Antenna–Rules Options...** subcommand.



[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 9: TOOLS



9-4: Simulation



The **Simulation (Built-in)** command of the **Tools** menu controls the internal simulator, which can simulate schematics, IC layout, and VHDL specifications. [Chapter 10](#) discusses this simulator more fully.

The **Simulation (SPICE)**, **Simulation (Verilog)**, and **Simulation (Others)** commands of the **Tools** menu are able to produce input specifications for a number of different simulators. All these commands work on the current cell and require that all named points be exports. It is also necessary to have power and ground exports.

The possible simulators include circuit-level (SPICE, Maxwell), switch-level (IRSIM, ESIM, RSIM, RNL, COSMOS, and MOSSIM), gate-level (ABEL-PAL), and functional (VERILOG, TEGAS, and SILOS). In addition, you can generate a deck for the FastHenry inductance analysis system.

To produce an input deck for any of these simulators, use the **Write XXX Deck** command (where **XXX** is the simulator's name). In addition to these simulators, an EDIF netlist can be produced with the **EDIF (Electronic Design Interchange Format)** subcommand of the **Export** command of the **File** menu.

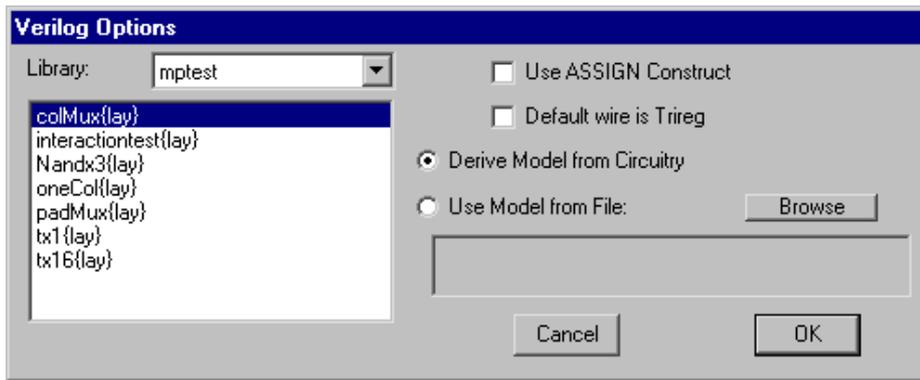
Verilog

Besides generating Verilog decks, it is possible to annotate circuits with additional Verilog declarations and code that will be included in the deck. The subcommands **Add Verilog Code** and **Add Verilog Declaration** of the **Simulation (Verilog)** command of the **Tools** menu allow you to click in the circuit and type code or declarations. These pieces of text can be manipulated like any other text object (see [Section 6-8](#) on text).

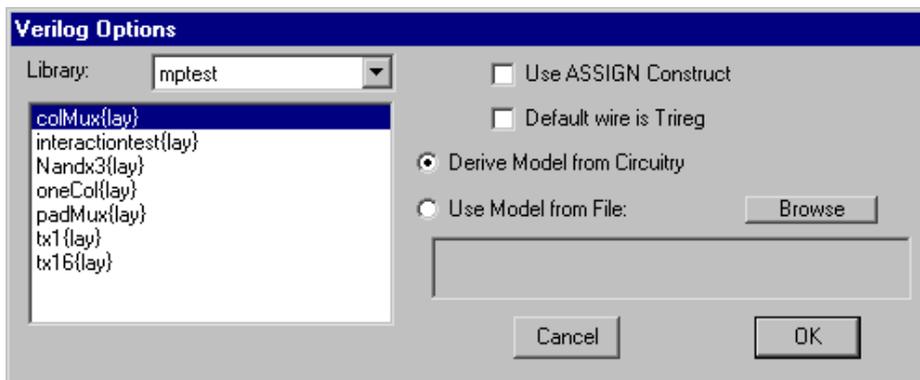
Additional control of Verilog deck generation is accomplished with the **Verilog Options...** subcommand. A checkbox lets you choose whether or not to use the Verilog "assign" construct. You can control the type of Verilog declaration that will be used for wires ("wire" by default, "triereg" if checked). Note that this can be overridden with the **Set Verilog Wire** subcommand of the **Simulation (Verilog)** command of the **Tools** menu.

The Verilog Options 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 library "samples.txt" (you can read the library with the **Readable Dump** subcommand of the **Import** command of the **File** menu).





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...** subcommand of the **Simulation (Verilog)** command of the **Tools** menu (or just **Plot Verilog for This Cell** if the cell name and file name are the same). Control of the waveform window is described more fully in [Section 10–2](#).



SPICE

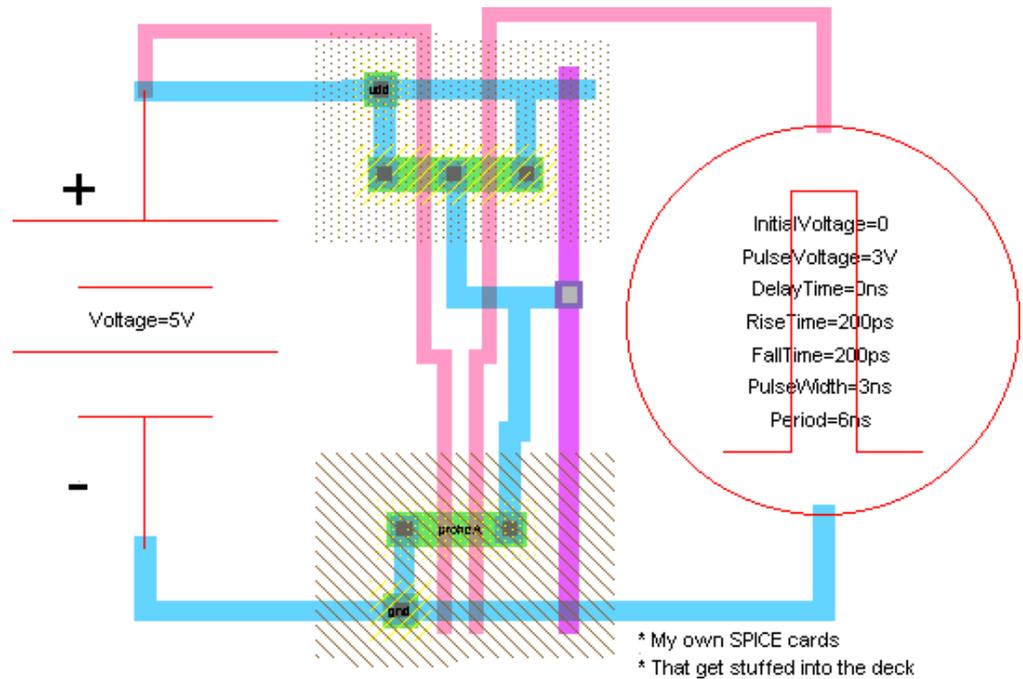
SPICE circuit simulation is a special case in Electric. Because the simulator is not interactive, all specifications must be done graphically, in advance. Note that the example shown here is available in the "samples.txt" library as cell "tool–SimulateSPICE" (you can read the library with the **Readable Dump** subcommand of the **Import** command of the **File** menu).



All input values to SPICE are controlled with special nodes, found in the **New SPICE**

Part command of the **Edit** menu.

These parts are also available from the "Spice" button in the component menu of the Schematics technologies. Note that the first time any SPICE node is placed, the library of SPICE parts is loaded into Electric.



The SPICE primitives described here are for Electric's default set. However, additional sets can (and have) been written. To choose another set, change the "SPICE primitive set" popup in the **SPICE Options...** subcommand of the **Simulation (SPICE)** command of the **Tools** menu.

In this example, there is a 5-volt supply on the left. It was created by using the **DC Voltage Source** subcommand. Once placed, the text that reads "Voltage=0V" is selected and modified (either with **Get Info** or by double-clicking on it). The Pulse input signal on the right is created with the **Pulse** subcommand (it has 7 parameters).

This example also shows the ability to add arbitrary text to the SPICE deck with the **Add SPICE Card** subcommand of the **Simulation (SPICE)** command of the **Tools** menu. The command creates a piece of text (shown in the lower-right) that can be modified arbitrarily.

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.

Bipolar transistors have their substrate connected to ground by default. If a **Substrate** node is encountered, its network will be used for the substrate connection of these transistors instead. The **Well** subcommand can be used to specify the well network.

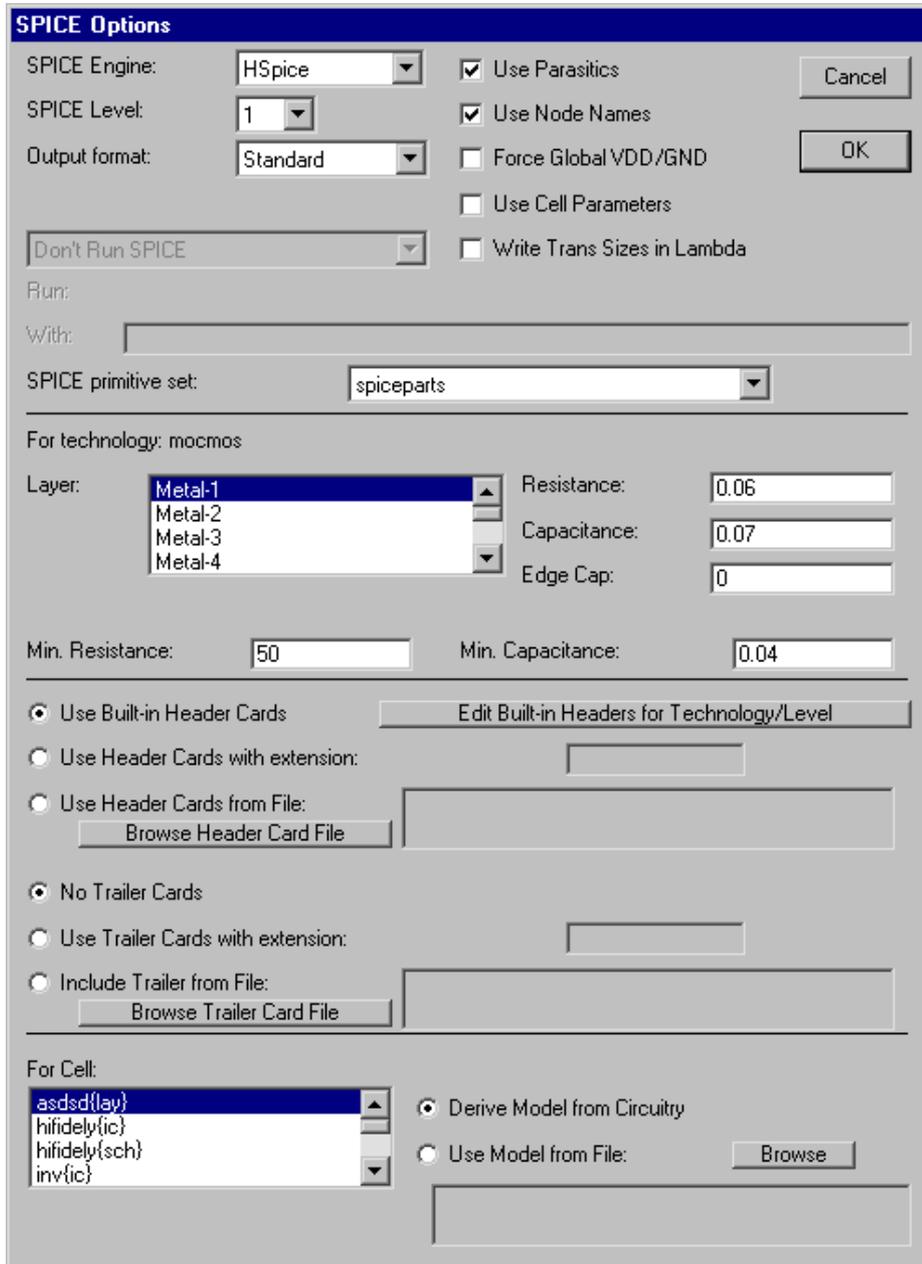
For advanced users, there are two special SPICE nodes: **Node Set** and **Extension**. The Node Set may be parameterized with an arbitrary piece of SPICE code. Truly advanced users may create their own SPICE nodes by modifying the cells in the SPICE library.

Another option that can be used when modeling transistors and other component is to set a specific SPICE model to use for that component. Use the **Set Spice Model...** subcommand of the **Simulation**



(**SPICE**) command of the **Tools** menu to create a SPICE model field on the selected component. Then you can select that text and set it.

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 Options...** subcommand of the **Simulation (SPICE)** command of the **Tools** menu.

The top part of this dialog allows you to control many of the SPICE deck parameters such as the SPICE format (SPICE 2, SPICE 3, HSPICE, PSPICE, Gnucap, or SmartSPICE), the SPICE level (1, 2, or 3), what format to expect when reading SPICE output, whether to use parasitics in the deck, whether to use actual node names in the deck (SPICE 2 cannot handle this), and whether to force power and ground to be global signal names.

When "Use Cell Parameters" is checked, any parameters defined on the cell will appear in the SPICE deck. When this is not checked, each parameterized cell appears multiple times in the deck, once for each different parameter combination. See [Section 6–8](#) for more on parameters.

The "Write Trans Sizes in Lambda" dialog requests that the SPICE deck contain scalable size information instead of absolute size information.



UNIX systems can choose to run SPICE after the deck has been generated. In addition, they can specify special command–line options to be given to SPICE (the "With" field). There are five options:

- "Don't Run SPICE" requests deck generation only.
- "Run SPICE" causes SPICE to be run after deck generation, and the output shown in the messages window.
- "Run SPICE Quietly", causes SPICE to be run after deck generation, but the output is ignored.
- "Run SPICE, Read Output" causes SPICE to be run after deck generation, output to be shown in the messages window, and the output to be examined for waveform values (which are then shown in a waveform window).
- "Run SPICE Quietly, Read Output" causes SPICE to be run after deck generation, output to be not shown, and the output to be examined for waveform values (which are then shown in a waveform window).

Note that SPICE 2 and SPICE 3 place their waveform values in the output, but HSPICE writes waveform values to a disk file (.tr0 file). Therefore, it does not make sense to choose "Run SPICE, Read Output" or "Run SPICE Quietly, Read Output" when using HSPICE.

The middle section of the Spice Options dialog controls technology–specific information. The upper–middle section controls parasitics, including per–layer resistance, capacitance, and edge–capacitance. You can also set the minimum resistance and capacitance for the entire technology. The lower–middle section controls header cards (placed at the start of the SPICE deck) and trailer cards (placed at the end of the SPICE deck). This dialog allows you to specify a disk file with header cards or trailer 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. You can also edit the built–in header cards for the current technology by using the "Edit Built–in Header Cards" button, which invokes an editing window (see [Section 4–10](#) for more on text editing).

Note that the header, trailer, and parasitic 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 **Technology Options** command of the **Technology** menu and setting the "Use Lambda values from this Technology" popup.

The bottom section of the dialog allows you to specify a disk file of SPICE cards that will be used to describe any cell. This disk file replaces the any SPICE description that may be derived from the circuitry.

SPICE and Verilog Primitives

Electric has a set of SPICE elements (DC Voltage Source, CCCS, Pulse, etc.) that create appropriate SPICE deck cards. These elements are found in the readable dump file "spiceparts.txt" in the "lib" directory. This library consists of a set of icon cells that describe the various SPICE primitives, and each icon has a special *Spice Template* that describes the SPICE card to generate.

Users can define their own SPICE elements by creating new icons in this or a new library. The icon must have graphics, exports, parameters, and a template. Parameters are created with the **Cell Parameters...** subcommand of the **Attributes** command of the **Info** menu (see [Section 6–8](#) for more on parameters). The SPICE template is created with the **Set Generic SPICE Template** subcommand of the **Simulation (SPICE)** command of the **Tools** menu. If the template is specific to a particular version of SPICE, use the appropriate template subcommand (**Set SPICE 2 Template**, **Set SPICE 3 Template**, **Set HSPICE Template**, **Set PSPICE Template**, **Set GnuCAP Template**, or **Set SmartSPICE Template**).

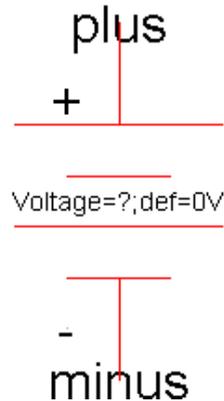


You can also create Verilog elements by using the **Set Verilog Template** subcommand of the **Simulation (Verilog)** command of the **Tools** menu. This template has the format as illustrated below. Note that a single cell can contain both Verilog and multiple SPICE templates.

The DC Voltage Source primitive is illustrated here. 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

```
SPICE_template=V$(node_name) $(plus) $(minus) DC $(Voltage)
```

```
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; and $\$(Voltage)$ will be replaced with the voltage value specified by the user.

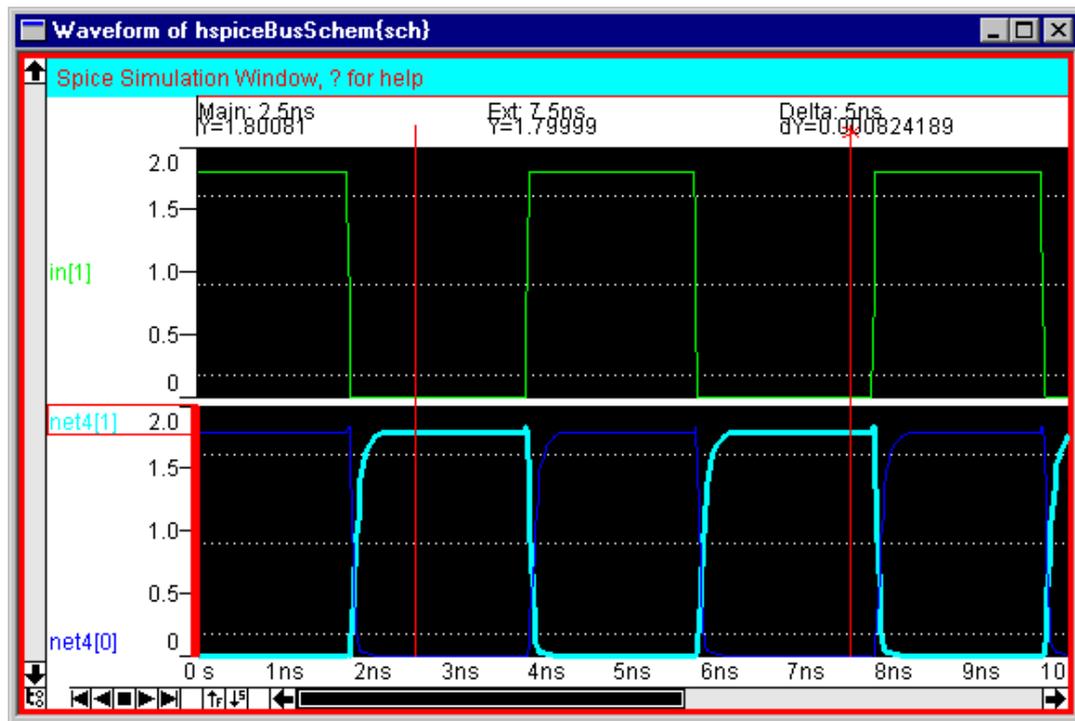
The set of SPICE primitives in Electric is useful, but far from complete. A second set, called "SpicePartsS3", is tailored towards special SPICE3 primitives. There are no Verilog primitives in the current release of Electric. Users who define new primitives are encouraged to share these with the entire community by contacting [Static Free Software](#).

SPICE Plotting

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 many formats of SPICE output, you must first set the "SPICE Engine" and the "Output format" fields of the **SPICE Options...** dialog. The "Output format" field can be "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 to read the file (or just **Plot Spice for This Cell** if the cell name and file name are the same). The waveforms will appear in a window:



The waveform window is tied to an associated schematics or layout window. Clicking on a signal in the waveform windows highlights the equivalent circuitry in the other window, and clicking on an arc in the schematics or layout window causes the waveform signal to be selected. Both the waveform window and the associated schematics/layout window are highlighted with a red border to indicate that they are part of the simulation activity.

The horizontal axis of the simulation window shows time. Two vertical lines are drawn, called the "main" and the "extension" cursors (the extension cursor has an "X" drawn at its top). You can click over these cursors and drag them to different time locations. The location of the cursors, and the value of the selected signal at those times, is shown at the top. The time axis can also be controlled with the appropriate **Windows** menu commands. Use **Zoom Out** and **Zoom In** to scale the time axis by a factor of two. Use **Fill Window** to display the entire range of data, fit to the screen. Use **Focus on Highlighted** to display the range between the main and extension cursors.

You can control which signals are displayed in the waveform window. To remove a signal, select its name and type "r" or the DELETE key. When adding a signal, you have a choice of showing it overlaid with an existing signal, or in its own graph. Typing "o" causes the signal to be overlaid onto the currently selected signal, and typing "a" causes the signal to be added in its own "frame". If you type "o" or "a" into the waveform window, you will be prompted for a list of signal names to display. If you type "o" or "a" in the associated schematics/layout window, then the selected signal from that window will be added to the waveform display.

Besides the time axis, it is possible to zoom and pan the vertical axis of a frame of the waveform window. Typing '7' doubles the scale (zooms-in) and typing '0' halves the scale (zooms-out). Type '9' to restore the scale so that the data fills the screen. Use '8' and '2' to shift the data values up and down.



One final feature is the ability to take a snapshot of the waveform window. Typing "p" preserves the waveform in the database (a cell with the "simulation-snapshot" view is created with artwork components).

Here is a summary of the single-key commands available in SPICE windows:

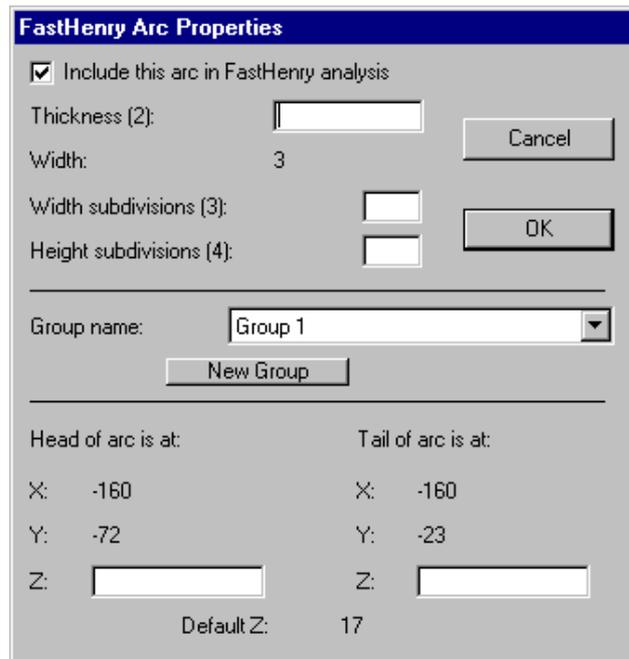
Key	Waveform Window	Schematics/Layout Window
a	Add signal	Add selected network to waveform window
o	Overlay signal on top of currently selected signal line	Overlay selected network on top of currently selected signal line
r DEL	Remove selected signal	
7	Zoom in vertically (double data scale)	
0	Zoom out vertically (halve data scale)	
9	Scale data to fill the display	
8	Shift data up by 1/4 screen	
2	Shift data down by 1/4 screen	
p	Preserve snapshot of waveform window	
d		Move down the hierarchy (into the selected cell)
u		Move up the hierarchy (out of the current cell)
?	Print this listing of single-key commands	Print this listing of single-key commands

FashHenry

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 Info...** subcommand of the **Simulation (Others)** command of the **Tools** menu.

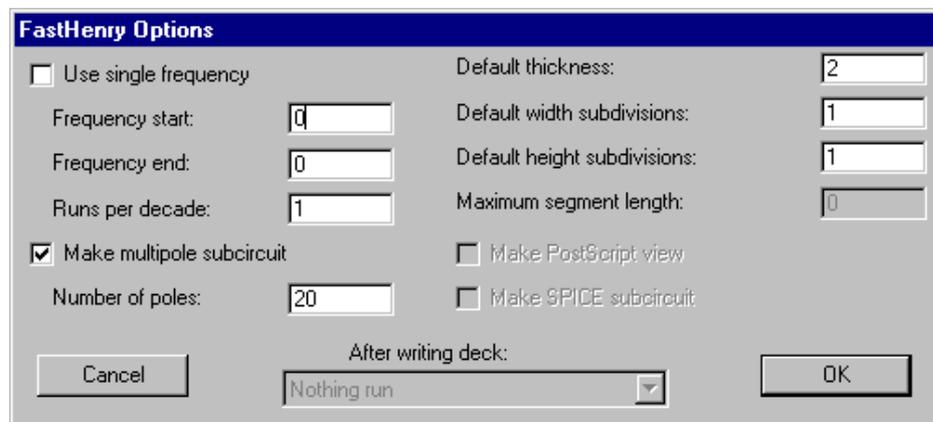


This command presents a dialog with FastHenry factors for the selected arc. The most important factor is in the upper-left: "Include this arc in FastHenry analysis". By checking this, the arc is described in the FastHenry deck. Once this is checked, other fields in the dialog become active. You can set 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.



You can partition the arcs into different groups. By default, all arcs are placed in a group called "Group 1". By clicking the "New Group" button, you can type a new group name, and that group can be used in some of the arcs.

After all arcs have been marked, you can generate a FastHenry deck with the **Write FastHenry Deck...** subcommand of the **Simulation (Others)** command of the **Tools** menu. Before doing that, however, you can set other options for FastHenry deck generation. To do this, use the **FastHenry Options...** subcommand of the **Simulation (Others)** command of the **Tools** menu.



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 frequency). 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). Other options are not implemented at this time.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 9: TOOLS



9-5: 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 7-9](#)). To create one, use the **Get Unrouted Wire** subcommand of the **Routing** command of the **Tools** menu before making a connection. Another way to get unrouted wires is to select all or part of an existing route (made with any arc) and use the **Unroute** subcommand.

Finally, the **Copy Routing Topology** and **Paste Routing Topology** subcommands 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.

Auto Stitching

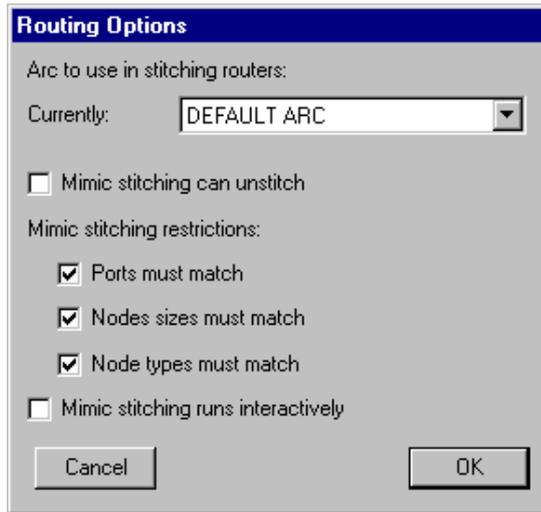
The auto-stitching router looks for adjoining nodes that make implicit connections, and places wires at those connections to make them explicit. For example, if a cell has power and ground rails at the top and bottom, and there are ports on the left and right of each rail, then the auto-stitching router 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** subcommand of the **Routing** command of the **Tools** menu. The router will make all necessary connections, and incrementally add wires as further changes are made to the circuit. To stop stitching, use the same menu entry, which now reads **Disable Auto-Stitching**. To run the auto-stitcher only once, and in the highlighted area only, use the **Auto-Stitch Highlighted Now** subcommand. 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.





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, with the **Routing Options...** subcommand, a specified wire can be given (or automatic selection can be resumed by selecting the "DEFAULT ARC" entry).

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 cannot add all necessary wires.

To control the wiring of arrays of cells more directly, there is the mimic-router. 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 single wires (if you make a multibend wire, that cannot be mimicked) in all situations where the same ports on the same type of nodes exist, separated by the same distance.

The **Routing Options...** command provides variations on these rules. First, you can request that the mimic stitcher also mimic wire deletions. Second, you can request that the mimic stitcher relax its restriction about mimicking 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** subcommand, and to disable the subsystem, use the command in the same menu location, which now reads **Disable Mimic-Stitching**. 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.

Maze Routing

The maze router replaces unrouted arcs with actual geometry. The **Maze-Route Selected** subcommand replaces the selected unrouted arcs, and the **Maze-Route Cell** subcommand replaces all unrouted arcs in the cell.

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 circuit, 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 library file "samples.txt" and edit the cell "tool-RoutingMaze" (you can read the library with the **Readable Dump** subcommand of the **Import** command of the **File** menu). This cell has a number of unrouted wires that can be routed.

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** subcommand of the **Routing** command of the **Tools** menu. 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 library file "samples.txt" and edit the cell "tool-RoutingRiver" (you can read the library with the **Readable Dump** subcommand of the **Import** command of the **File** menu).

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 9: TOOLS



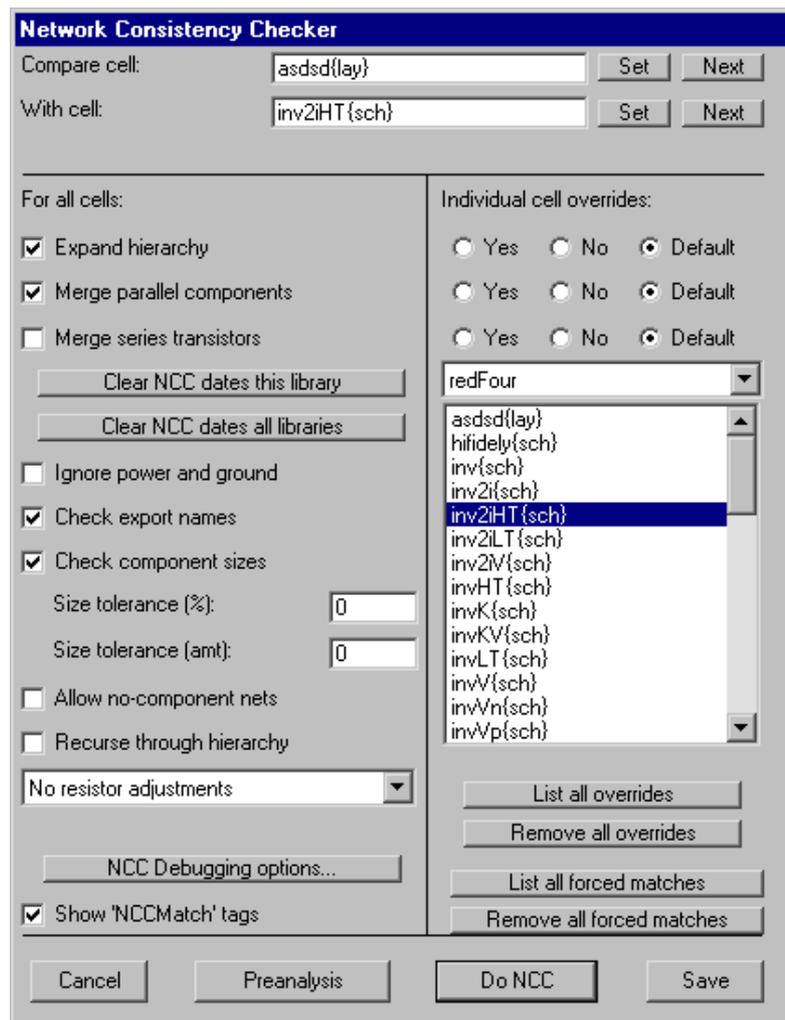
9-6: Network Consistency Checking (NCC, or LVS)



Network Comparison

Electric is able to 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) the term Network Consistency Checking (NCC) is used. Electric's NCC system is based on the [Gemini](#) work of Carl Ebeling (see Ebeling, Carl, "GeminiII: A Second Generation Layout Validation Program", *Proceedings of ICCAD 1988*, p322-325.)

To compare two cells, use the **NCC Control and Options...** subcommand of the **Network** command of the **Tools** menu.



The top part of the dialog lists the two cells that are to be compared. If two different cells are currently being displayed on the screen, they are loaded into the dialog. Otherwise, you can select the cells with the "Set"



buttons. If there are many cells being displayed, the "Next" buttons will cycle through them. After the first use of this dialog, it remembers the last two cells that were compared.

The bottom of the dialog has buttons for running NCC ("Do NCC") and for doing a "Preanalysis" (a quick comparison of the cells that can discover comparison problems without doing the full analysis). The preanalysis shows networks and components in the two cells and lets you examine those that are different. The "Save" button saves the changes made in the dialog without doing any analysis.

For an example of network consistency checking, open the library "samples.txt" and compare the cells "tool-NCC{lay}" and "tool-NCC{sch}" (you can read the library with the **Readable Dump** subcommand of the **Import** command of the **File** menu). These two cells are equivalent and the checker will find them to be so.

When you request comparison, the system displays the number of components, networks, and other information in each cell. Inconsistencies in these numbers generally lead to failure of comparison. If inconsistencies are found, you will be asked if you want to stop, do a Preanalysis, or continue with the full NCC.

When comparison fails, you can review the errors by typing ">" and "<" to step to the next and previous error that was found. This list is also available in the Cell Explorer (see [Section 3-7](#)).

Once compared successfully, nodes and arcs in one cell can be matched to those in the other simply by selecting one and using the **Show Network** subcommand.

Fine-Tuning

To control the network consistency checker, use the center portion of the **NCC Control and Options...** dialog. The options on the left are defaults for all cells. Some of these options can be overridden for individual cells by selecting that cell and choosing the "Yes" or "No" override buttons on the right. Note that overrides apply to schematics and not their icons (icon cells are not shown in the list). To see a list of all overrides that exist, use the "List all overrides" button.

For an initial comparison of two cells, it is best to leave all options off. The system can automatically detect some of the options, and you can choose to select others as needed.

When checking a particular cell, the NCC can ignore or examine the contents of instances. If "Expand hierarchy" is checked, then all circuitry below the current cell is extracted and considered with the cell. Otherwise, cell instances are only compared by their connections without regard to their contents. To help align two hierarchies that are structured differently, this option can be applied selectively to different cells.

The "Merge parallel components" checkbox instructs the NCC to consider multiple components wired in parallel to be a single component. When this option is selected, two or more parallel components will correctly match a single, larger component in the other cell.

The "Merge series transistors" checkbox instructs the NCC to consider multiple transistors wired in series to be a single, complex component. When this option is selected, the order of wiring gates to these transistors will not be ignored.

The NCC remembers the time at which a successful NCC was done and marks the matched cells so that they are not checked again (if they haven't changed). The "Clear NCC dates this library" button removes this information from the current library, forcing the NCC to run again. The "Clear NCC dates all libraries" button does the same thing for every library. Note that these buttons also remove "NCCmatch" tags that are



created by the matching process.

These forced matches are created automatically during NCC and may also be placed by the user, using the **Create NCC Forced Association** subcommand of the **Network** command of the **Tools** menu. The command places an NCC-match tag on the currently select object. By changing the tag's name, and setting that name on objects in different cells, those objects are forced to match during NCC. To review these forced-matches, use the "List all forced matches" button; to clear them without clearing valid NCC dates, use the "Remove all forced matches" button.

"Ignore power and ground" instructs the NCC to ignore all power and ground networks. This is useful when the layout has power and ground but the schematic doesn't.

The "Check export names" check instructs the NCC to check export names for consistency after a match is found. If exports are named differently in the two circuits, warnings will be issued.

"Check component sizes" instructs the NCC to compare component sizes after a match is found. Unless this is checked, component will be equated only according to their connectivity, and not their size. The size tolerance fields allow slop in the percentage and absolute difference between two components.

The system will warn about networks that do not connect to any components unless "Allow no-component nets" is checked.

The network consistency checker can work on the current cell, or it can recursively check each cell from the current point on downward. To recursive check individual levels of the hierarchy, uncheck the "Expand hierarchy" box and check the "Recurse through hierarchy" box.

There is a popup that allows you to control how resistors will be treated. Resistors can be included or excluded from circuit analysis (see XXX). NCC typically wants to exclude resistors. The popup lets you choose 3 actions that can occur: (1) "No resistor adjustment" causes the current state of resistor inclusion/exclusion to remains for an NCC run; (2) "Include Resistors" forces resistors to be included before an NCC run; and (3) "Exclude Resistors" forces resistors to be ignored before an NCC run.

The "NCC Debugging options..." button brings up a dialog for helping to debug this system. It is not intended for users and is not discussed further.

Disambiguation

During comparison, there are often situations where a group of networks or components from one cell are equivalent to a group in the other cell. Because they are structurally ambiguous, the NCC tries to disambiguate them and achieve a complete match. The first thing that is checked is names (export names, network names, or node names).

Other techniques for disambiguation include node sizes, and random guessing. When a random guess is made, tags are placed in the circuit to show what was presumed. These labels have names like "NCCMatch3" and are placed on nodes or arcs to indicate presumed association. The "Show 'NCCMatch' Tags" requests that these automatically-generated equivalence markers be displayed in the circuit.

[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 9: TOOLS



9-7: PLA and ROM Generation



Introduction to PLAs

PLA generation is a process by which a set of input signals combines, through a logical sum of products, to form a set of output signals. For example, there may be two outputs: f and g , which are defined as follows:

$$f = (a \text{ and } b \text{ and } (\text{not } c)) \text{ or } ((\text{not } b) \text{ and } (\text{not } a))$$
$$g = (a \text{ and } c) \text{ or } ((\text{not } a) \text{ and } (\text{not } c))$$

This is a logical sum (**or**) of products (**and**), and the input terms may be negated (**not**). PLA generators require this information in the form of two personality tables: an AND table and an OR table. The AND table is as wide as there are inputs (3 in this case), and the OR table is as wide as there are outputs (2 in this case). The height of the tables is determined by the number of "product terms," which are the number of intermediate results required to define the logic (4 in this case). The AND table for the above equations is:

a	b	c	
1	1	0	$a \text{ and } b \text{ and } (\text{not } c)$
0	0	X	$(\text{not } b) \text{ and } (\text{not } a)$
1	X	1	$(a \text{ and } c)$
0	X	0	$(\text{not } a) \text{ and } (\text{not } c)$

Notice that there is a "1" where the input term is in a positive form, a "0" where the input term is in a negated form, and an "X" where the input term does not apply. The OR table for the above equations then combines the four product terms into the two output terms as follows:

f	g	
1	0	$f: a \text{ and } b \text{ and } (\text{not } c)$
1	0	$f: (\text{not } b) \text{ and } (\text{not } a)$
0	1	$g: (a \text{ and } c)$
0	1	$g: (\text{not } a) \text{ and } (\text{not } c)$

Electric's PLA generator tool consists of two different generators: an nMOS generator and a CMOS generator. Both use personality tables to specify which taps in the programming array are set. Both produce a hierarchical array specification made up of AND tables, OR tables, drivers, and all necessary power and ground wires.



The nMOS PLA Generator

The nMOS generator produces a circuit in the "nmos" technology. The PLA is generated with the **Make nMOS PLA** subcommand of the **PLA Generator** command of the **Tools** menu. You will be prompted for the file name that describes the PLA.

Below is a sample file which defines the above logic as an nMOS PLA (this file can be found in the **PLA-ROM** subdirectory of the **examples** directory). Note that comments can be inserted after a semicolon. The number of inputs, outputs, and product terms must be provided so that the array of values between the "begin" and "end" can be properly parsed. The other parameters are optional. These include the power and ground widths (default is 4 lambda); whether to use butting-contacts or buried contacts (default is to use butting contacts); whether the outputs are on the same side as the inputs (default is to place on the opposite side); what constraints will be placed on the arcs in the PLA (default is nonrigid fixed-angle); and a name for the newly created PLA cell (default is "nmosXXX" where "XXX" is the PLA size).

```
set inputs = 3                ; sum of input and output is
set outputs = 2              ; number of columns
set pterms = 4               ; 4 product terms (number of rows)
set vddwidth = 6             ; 6 lambda-wide supply rails
set groundwidth = 6
set buttingcontact = off     ; use buried contacts instead
set samesideoutput = on      ; outputs on same side as inputs
set flexible = on            ; use nonrigid arcs
set fixedangle = on          ; use fixed-angle arcs
set name = Sample            ; name to use for top-level cell
begin ; Input      Output
      ;  1  2  3      1  2
      1  1  0      1  0   ; product term 1
      0  0  X      1  0   ; product term 2
      1  X  1      0  1   ; product term 3
      0  X  0      0  1   ; product term 4
end
```

The CMOS PLA Generator

The CMOS PLA generator is somewhat more flexible than the nMOS version because it reads a library of support cells and uses them to produce the array. This means that it can handle any technology (although the only library that comes with Electric is for the MOSIS CMOS technology). For those who wish to construct their own library in another technology, note that it must contain the cells "decoder_inv1", "io-inv-4", "nmos_one", "pmos_one" and "pullups". Look at the library "pla_mocmos" (in the **lib** directory) for more information.

The CMOS PLA generator is run with the **Make MOSIS CMOS PLA** subcommand of the **PLA Generator** command of the **Tools** menu. You are then prompted for two files: the AND table file and the OR table file. These files are much simpler in format than the nMOS PLA input file. They have 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 the AND file for the above logic:

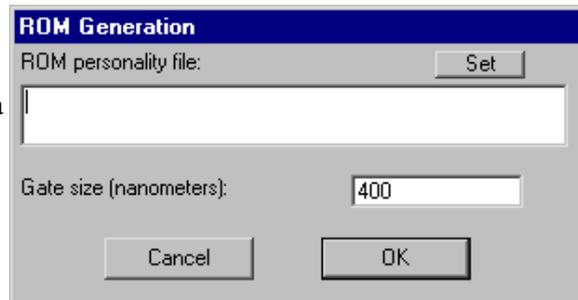
```
4  3
1  1  0
0  0  X
1  X  1
0  X  0
```



The ROM Generator

The ROM generator reads a single personality table and builds a ROM. Since the generator is written in Java, you must have Java installed in Electric in order for this to run.

The first line of the ROM personality table 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.



For an example of a ROM personality table, see the file **rom.txt** in the **PLA-ROM** subdirectory of the **examples** directory.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 9: TOOLS



9–8: 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
facet PADFRAMECELL                       ; Creates a cell to hold the pad frame
core CORECELL                             ; Places your circuit in the center of the pad
                                           ; frame
align PADCELL INPUTPORT OUTPUTPORT       ; Defines input and output ports on 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 lambda 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 tap=o7
```

creates an export on the pad from its "tap" port and names it "o7".

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. The array file is "pads4u.arr" (from the "Examples" folder) and it expects to find a cell called "tool-PadFrame" (found in the "samples.txt" library, which you can read with the **Readable Dump** subcommand of the **Import** command of the **File** menu).

```
; specify the library with the pads
celllibrary pads4u.txt

; create a cell called "padframe"
facet 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

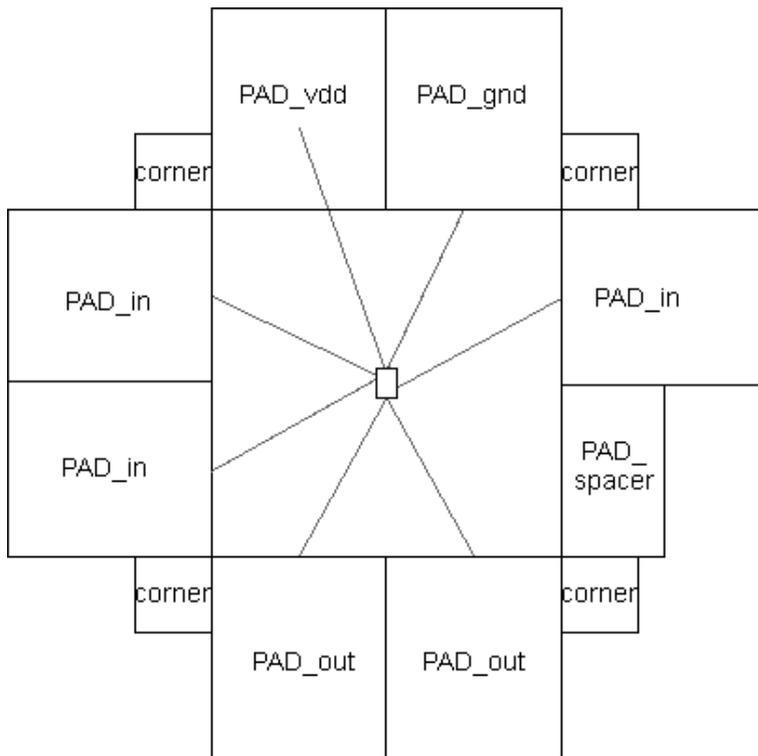
; place the top edge of pads
place PAD_corner{lay}
place PAD_gnd{lay} gnd_in=gnd
place PAD_vdd{lay} m1m2=vdd

; 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
```

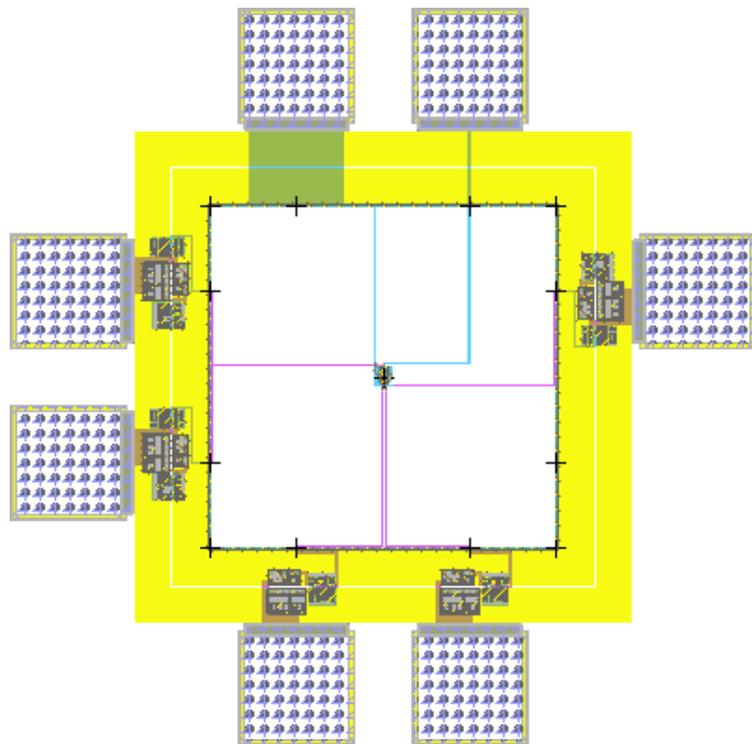




This file places 8 pads in a ring (2 on each side) and also places corner "pads" for making bends. The input pads connect to the 2 input ports "a1" and "a2". The output pads connect to the 3 output ports "out1", "out2", and "out3". The power and ground pads connect to the "vdd" and "gnd" ports.

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 9-5](#)).

Connections between pads and ports of the core cell use Unrouted arcs (from the Generic technology, see [Section 7-9](#)). These arcs can be converted to real geometry with the river router. To do this, you must select arcs on one side of the pad frame and use the **River-Route** subcommand of the **Routing** command of the **Tools** menu (see [Section 9-5](#) for more on routing). Because the river router always pushes geometry to the left and bottom, this will work for the left and bottom sides only. To route the top and right sides, you must rotate the entire circuit (select everything and rotate 180 degrees). After routing the top and right (now left and bottom) you can rotate the circuit back to its original position. The finished layout is shown here, fully instantiated.



[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 9: TOOLS



9–9: 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 structural VHDL description. The VHDL is compiled into a netlist which is then used to drive placement and routing. Also, because Electric is able to generate VHDL from a schematic, the silicon compiler can hide the VHDL and produce layout directly from schematics.

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 3 and 4 metal processes to route over the standard cells, but this system does not.

The VHDL description is normally placed in the "vhdl" view of a cell. It can be created and edited entirely inside of Electric, or it can be read from disk by using the **Read Text Cell...** command of the **Cells** menu. See [Section 4–10](#) for more on text editing. Automatic generation of VHDL from layout is done with the **Make VHDL View** command of the **View** menu.

Once the VHDL is created, it is compiled into a netlist and read into the silicon compiler. The netlist is normally placed in the "netlist–quisc–format" view of a cell. If the netlist is too cumbersome to retain in memory, it may be kept on disk by using the **VHDL Options...** subcommand of the **VHDL Compiler** command of the **Tools** menu and unchecking the "Netlist stored in cell" item. The "VHDL stored in cell" check controls the placement of VHDL text in a similar way.

When generating a schematic or VHDL description, 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;

To use the silicon compiler, simply run the subcommands in the **Silicon Compiler** command of the **Tools** menu. The commands are organized in the menu so that, when run sequentially downward, they perform the compilation process. The steps are as follows:

- Create a circuit to be compiled (there is a VHDL example in cell "tool-SiliconCompiler" of the "samples.txt" library, and you can read the library with the **Readable Dump** subcommand of the **Import** command of the **File** menu).
- Use the **Read MOSIS CMOS Library** subcommand to read the cell library.
- Use the **Silicon Compiler Options...** subcommand to set parameters.

Because cells are laid out in rows that run horizontally, the Vertical routing arc runs in and out of cells, while the Horizontal routing arc runs between the cells in the routing channel. The Power arcs run horizontally between the cells to connect both power and ground. The Main Power arcs run vertically along the sides of the circuit to connect the horizontal power and ground rails (you can choose which layer to use for the main power arcs). A block of P-well will be placed along the bottom of each cell and extend up to the P-Well height (if nonzero). A block of N-well will be placed along the top of each cell and extend down to the N-Well height (if nonzero). The offset of these blocks from the bottom or top can also be given. The Via size, Minimum metal spacing, Routing feed-through size, Minimum port distance, and Minimum active distance are rules that are used to place wires in the routing channel.

Parameter	Value
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:	Horizontal Arc
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
Via size:	4
Minimum metal spacing:	6
Routing: feed-through size:	16
Routing: min. port distance:	8
Routing: min. active dist.:	8
Number of rows of cells:	4



Finally, the Number of rows of cells specifies how many rows to use when creating layout. 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).

- Use the **Get Network for Current Cell** subcommand to get the netlist associated with the current cell. This command will compile VHDL if the netlist is unavailable or out of date (note, however, that if the current cell is a netlist, it will be used without consideration of more recent VHDL or layout cells). This command will also generate VHDL from a layout if the VHDL is missing or out of date (note also that if the current cell is VHDL, it will be used without consideration of more recent layout cells).
- Use the **Do Placement** subcommand to place the library cells.
- Use the **Do Routing** subcommand to wire them together.
- Use the **Make Electric Layout** subcommand to actually create the placed–and–routed circuit in the current library. The incremental design–rule checker is turned off at this point because the new cell will be vast and may be troublesome. The new cell is given the "layout" view.

The last subcommand in the **Silicon Compiler** command is **Issue Special Instructions...**, which does not normally need to be used. However, the silicon compiler system is so extensive that advanced users may wish to use it. After issuing this command, any sequence of silicon compiler instructions may be typed. Use the "Cancel" button to return to Electric.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 9: TOOLS



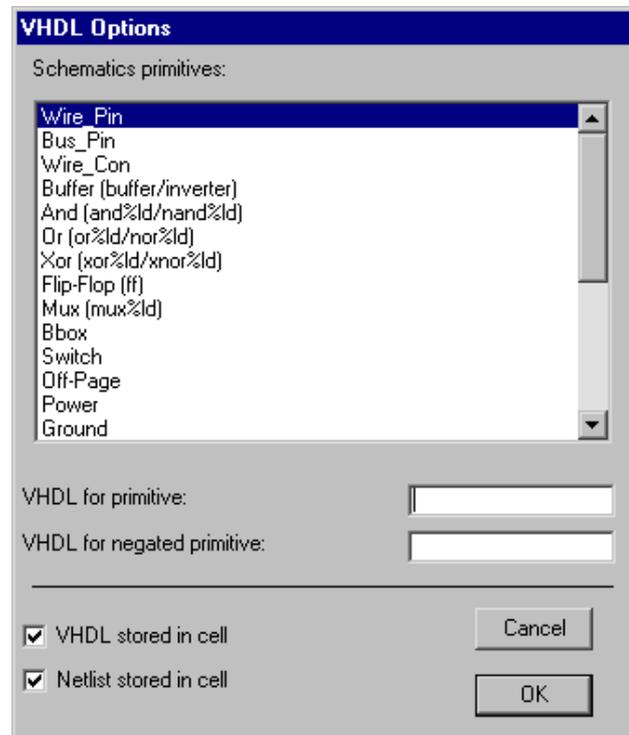
9-10: VHDL Compiler



The VHDL compiler translates structural VHDL code into netlists. It can generate a number of different netlist formats including ones for the simulator and the silicon compiler. Typical use of the simulator and silicon compiler automatically drives the VHDL compiler, so the average user will not need to use the commands described here.

Use the **Compile for Silicon Compiler**, **Compile for Simulation**, **Compile for RNL**, **Compile for RSIM**, and **Compile for SILOS** subcommands of the **VHDL Compiler** command of the **Tools** menu to generate appropriate format netlists.

By default, the compiler reads VHDL from the "vhdl" view of the cell in the current window and writes netlists to appropriate "netlist" views of this cell. This can be changed with the **VHDL Options...** subcommand of the **VHDL Compiler** command of the **Tools** menu. By unchecking "VHDL stored in cell", the VHDL is taken from the file "XXX.vhdl", where XXX is the current cell name. By unchecking "Netlist stored in cell", the netlist is written to the file "XXX.sci" (for the silicon compiler), "XXX.net" (for the simulator, RNL, and RSIM), or "XXX.sil" (for SILOS) where XXX is the current cell name. Recheck these items to locate the text in the cells.



The Options dialog also controls how the VHDL generator produces symbol names for schematics nodes. The dialog shows each schematics node, along with its regular and negated VHDL symbol (the use of "%d" is replaced by the number of inputs on the gate).

Another feature of the VHDL compiler is its ability to generate VHDL from a schematic or layout. Use the **Make VHDL View** command of the **View** menu to convert the current cell into VHDL. Note that the state of the "VHDL stored in cell" entry of the Options dialog determines whether this VHDL is written to disk or a



cell.

 Previous

 Table of
Contents

Next 



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** subcommand of the **Compaction** command of the **Tools** menu. You can also request that it do a single horizontal or vertical squeeze with the **Compact Horizontally** and **Compact Vertically** commands.

The **Compaction Options...** subcommand presents a dialog in which you can tell the compactor to expand the circuit if it is too close for the design rules. You can also request extensive information about the compaction process.



An example of compaction can be found in the "samples.txt" library (you can read the library with the **Readable Dump** subcommand of the **File** menu). Edit the cell "tool-Compaction" and compact it.

Be warned that the compaction tool is experimental and doesn't always achieve optimal results.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 9: TOOLS



9–12: 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).

When the Logical Effort tool runs, it annotates each digital schematic gate with a fanout ratio that can be used to size the transistors inside of that gate. It is up to the designer (or potentially some other tool) to use this information in the actual IC layout.

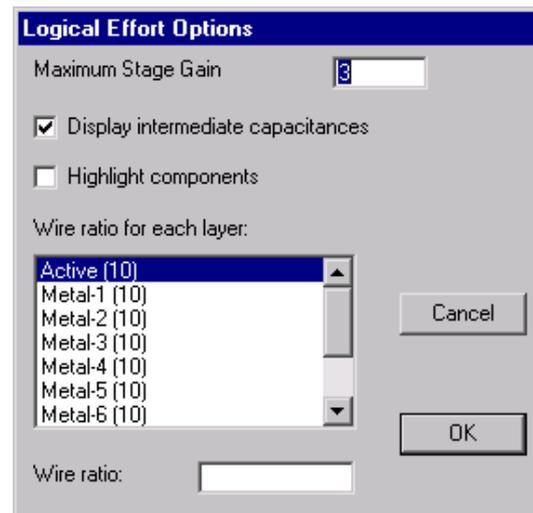
The Logical Effort tool has two functions: *path analysis* and *whole–cell analysis*.

In path analysis, the user must select two points in the circuit that define the ends of a path. Selection of these two points is done by clicking the *selection* button over the first point, and then using the *toggle select* button on the second. Capacitive loading on the ends of the path can be specified by creating Load objects and parameterizing their capacitance. These Load objects are created with the **New Arc Load** subcommand of the **Logical Effort** command of the **Tools** menu. To do the path analysis, use the **Analyze Path** subcommand, which determines the optimal fanout for each step in the path. Besides marking each gate with a fanout value (shown in the form of "h=2.5"), intermediate capacitance values are displayed along the path.

In whole–cell analysis, the Logical Effort tool iteratively applies capacitive loading constraints until the circuit fanout is determined. Once again capacitances can be set with the **New Arc Load** subcommand of the **Logical Effort** command of the **Tools** menu. Then, the **Analyze Cell** subcommand iterates over the circuit, attempting to find ratios that are less than or equal to the maximum stage gain (initially 3). As with path analysis, each gate is marked with a fanout value, and intermediate capacitances are displayed in the circuit. To change the maximum stage gain that is used in whole–cell analysis, use the **Logical Effort Options...** subcommand of the **Logical Effort** command of the **Tools** menu.



Besides setting the maximum stage gain for whole-cell analysis, the **Logical Effort Options...** subcommand of the **Logical Effort** command of the **Tools** menu allows other settings. You can request that intermediate capacitance values not be displayed after analysis, and you can request that the analyzed nodes be highlighted. Also, you set the wire ratio for each arc (a value used in load computation).



It is possible to override the default calculation of the logical effort on a single node by using the **Set Node Effort...** subcommand of the **Logical Effort** command of the **Tools** menu. The value entered is shown in the form "g=2" on the node, and is used in subsequent analysis.

An example of Logical Effort can be found in the "samples.txt" library (you can read the library with the **Readable Dump** subcommand of the **File** menu). Edit the cell "tool-LogicalEffort" and use the **Analyze Cell** subcommand on it.

There are three commands that analyze the circuit as an aid to doing Logical Effort. The **Estimate Delays** subcommand computes load factors for every network in the cell. This information is not generally useful, and is provided for future Logical Effort analyses. The **Show Network Loads** subcommand lists every network in the current cell, showing the wire length, load, and other information. The **Analyze Network** subcommand shows a detailed analysis of the currently selected network, including the area and perimeter information for each layer, as well as load information.

[← Previous](#) [↑ Table of Contents](#) [Next →](#)





Chapter 10: SIMULATION



10-1: Introduction to Simulation



Electric has a built-in gate-level simulator called ALS (the Asynchronous Logic Simulator) that can simulate schematics, IC layout, or VHDL descriptions. In addition, Electric can produce input decks for many external simulators. This chapter describes the built-in simulator. For more information about the simulation interfaces, see [Section 9-4](#).

For an example of simulation, open the library file "samples.txt" and edit the cell "tool-SimulateALS" (you can read the library with the **Readable Dump** subcommand of the **Import** command of the **File** menu). Then just issue the **Simulate...** subcommand of the **Simulation (Built-in)** command of the **Tools** menu.

A second simulation engine, Stanford's IRSIM, is also available for Electric. To obtain it, you must get the additional source code from [Static Free Software](#). Most operations that apply to ALS work the same with IRSIM. In the rest of this chapter, things specific to one of the simulators are marked so.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 10: SIMULATION



10-2: Simulator Operation



To begin simulation of the circuit in the current window, use the **Simulate...** subcommand of the **Simulation (Built-in)** command of the **Tools** menu. The simulator already knows about MOS transistors. The ALS simulator also knows about many digital logic gates and can be programmed to describe any function with the hardware description language described later in this chapter.

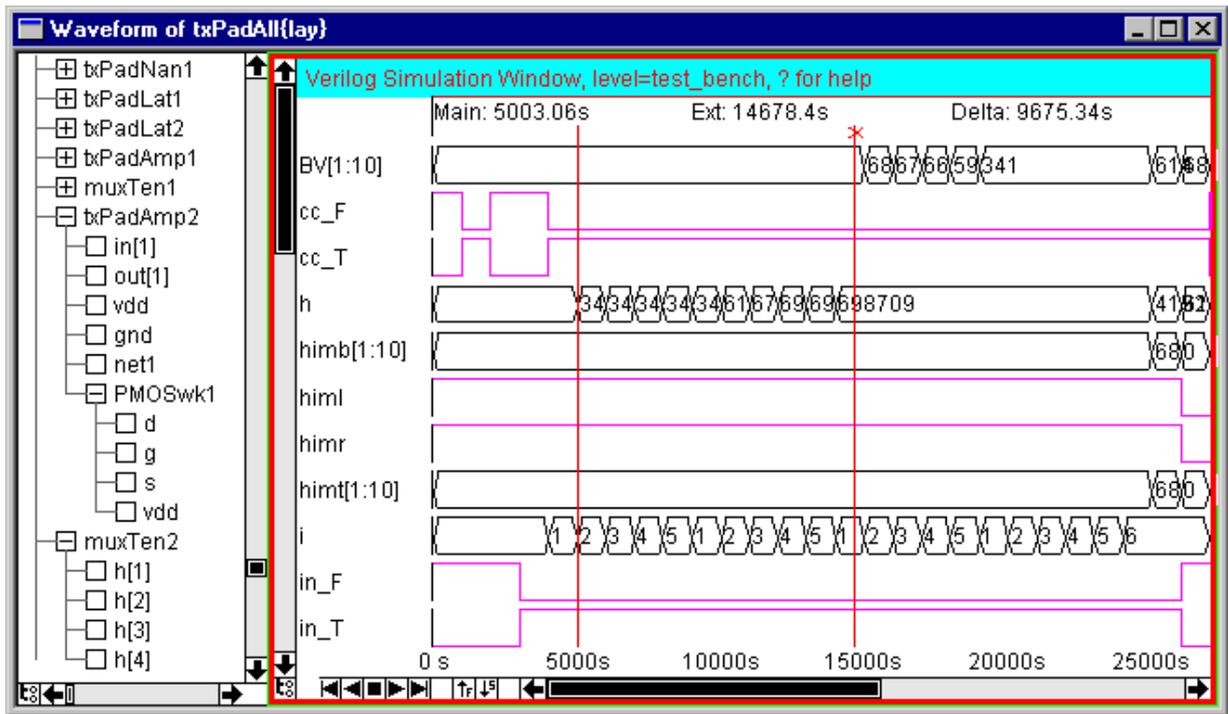
The Waveform Window

When simulation begins, a waveform appears in a new window which is highlighted with a red border. The circuit that is being simulated is also highlighted with a red border to indicate its association. Here is a sample waveform window when attached to the built-in ALS simulator:



This waveform window is also used to examine the results of batch simulations, such as SPICE and Verilog. Note that this Verilog output displays busses properly. Note also that the "explorer" panel (controlled by the tree icon in the lower-left corner) shows signals hierarchically.





In the main waveform window, Exports and other network names are listed on the left (if there are too many, then the slider allows them to be scrolled into view). The dividing line between signal names and signal values can be grabbed and moved to make room for wider signal names. You can click over a signal name or the actual waveform to select it for appropriate operations. 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 remove a signal by selecting it and typing the letter "r" (or the DELETE key). You can add a signal to the list by typing the letter "a" (you will then be prompted for the signal to add). If you are in the associated layout or schematics window and type "a", then that network is added. To aggregate a number of signals into a single "bus" signal, select them and type "b" (to select multiple signals, start in the blank space between signal names and drag vertically over multiple names). Once a bus signal has been created, its individual signals can be shown or hidden by double-clicking on its name. All of these rearrangements are remembered with the cell for subsequent simulations. With the ALS simulator, you can use the **Restore Signal Display Order (ALS)** subcommand to reset the waveform display to its original set of signals. With all simulators, you can use the **Remove Signals Saved with Cells** subcommand to clear the saved list of rearrangements.

Two vertical cursors appear in the window, called "main" and "extension" (the extension cursor has an "x" at the top). You can click over the cursors and drag them to different time locations. If one signal is selected, its value is shown at the main and extension times.

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. Use the **Cursor** subcommand of the **Center** command to shift the time to the location of the cursors.

The bottom of the waveform window has a set of VCR buttons that control an animation of the time. The Play rate can be controlled by the "F" and "S" buttons which make it go faster or slower. In addition, since the waveform window can light-up special "simulation probe" nodes, the VCR can be used to animate activity on a chip (see the section on "Simulator Control" below for more on this).



Test Vectors

When a signal name is selected, a test vector can be placed on that signal at the time specified by the main cursor. Type "l", "h" or "x" to set the signal to low, high, or undefined (using normal strength, shown in magenta). For bus signals, type "v" (you will then be prompted for the value to set on that bus). Prefix the letter with a "w" ("wl", "wh", "wx", and "wv") to set a weaker strength signal (shown in green). Prefix the letter with an "s" ("sl", "sh", "sx", and "sv") for stronger strength signals (shown in black).

Another way to control the strength of signals is by setting a transistor's strength. You can use the **Transistor Strength** subcommands of the **Simulation (Built-in)** command of the **Tools** window to control a selected transistor. Note that this must be done before simulation begins.

To remove the test vectors on the selected signal, type the "e" key. Use the **Clear All Vectors** subcommand of the **Simulation (Built-in)** command of the **Tools** menu to erase all test vectors. Note that these commands do not remove "strong" vectors (those set with the "sl", "sh", "sx", or "sv" commands).

Once vectors are established, the **Save Vectors to Disk** subcommand of the **Simulation (Built-in)** command of the **Tools** menu will write this information to disk. Use the **Restore Vectors from Disk** subcommand to read it back. You can even use the **Save Vectors as SPICE commands** subcommand to dump this information as a SPICE deck.

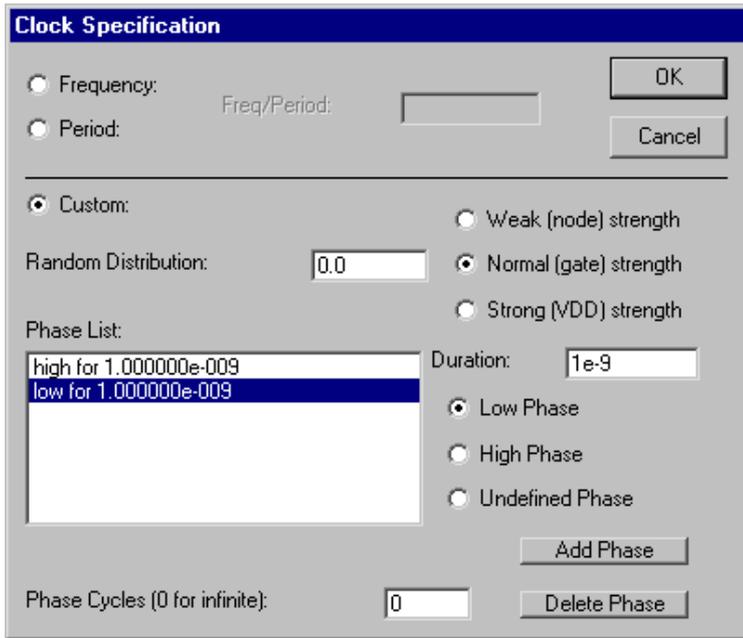
In the ALS simulator, you can also obtain simulation data from an SDF file (with the **SDF** subcommand of the **Import** command of the **File** menu). Once this data has been read, one of three sets of values (**Typical**, **Minimum**, or **Maximum**) must be selected from the **Annotate Delay Data (ALS)** subcommand of the **Simulate** command of the **Tools** menu.

Clocks

Besides simple test vectors, the ALS simulator can also set clock patterns on the currently selected signal by typing the letter "c". The user is then given a dialog for clock specification. Note that the clock may cycle 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.

There are three ways to specify a clock: by frequency, period, or with custom phases. If the frequency or period method is selected, the only option is the frequency (in cycles per second) or period (in seconds).





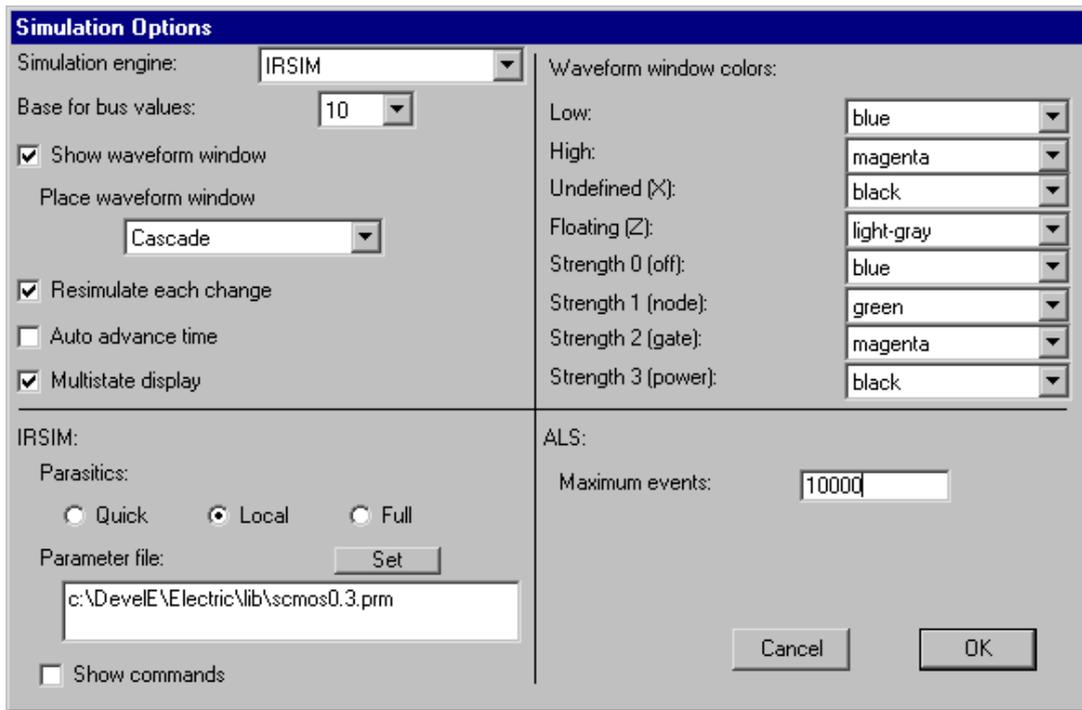
If, however, custom clock specifications are requested, the entire lower part of the dialog is enabled (as shown on the left) and many options are available. You can choose the strength of the clock (node, gate, or VDD), the random distribution of the clock, and a list of phases that will be repeated a specified number of times (use 0 repetitions for an infinite clock). For each phase, select its level (low, high, or undefined) and its duration. Then use the "Add Phase" button to add it to the list. Use the "Delete Phase" button to remove a phase.

Simulator Control

The simulator uses the hierarchy in the original circuit during simulation. Therefore, the signals shown are those from the current hierarchical level. If you use the **Down Hierarchy** and **Up Hierarchy** commands of the **Cells** menu to move through the hierarchy of the circuit, the waveform window will change to reflect the displayed circuitry.

The **Simulation Options...** subcommand of the **Simulation (Built-in)** command of the **Tools** menu offers control over the simulation process. Each change that is made to the simulator causes it to resimulate and display the results. If multiple changes are to be made and the simulation time is long, uncheck the "Resimulate each change" item. Then you must type "u" when you want to resimulate.





The "Auto advance time" check tells the simulator to move the time cursor automatically when a new signal is added to the simulation.

When simulating, the state of the circuit is displayed both in the waveform window and also on top of arcs in associated layout or schematics windows. The colors used can be controlled with choices in the upper-right part of the dialog.

The "Multistate display" check tells the simulator to show signals in the layout or schematics window with texturing and color to indicate strength. Without this, a simple on/off indication is drawn in the layout or schematics window. Keep in mind that the "Simulation Probe" node of the Generic technology also changes appearance to show simulation activity. You must connect it (with Universal arcs) to the part of the circuit that you want it to reflect. The Simulation Probe node is obtained by the **Simulation Probe** subcommand of the **New Special Object** command of the **Edit** menu.

The "Show waveform window" check tells the simulator to create a separate window with waveform plots when simulation starts. When this is unchecked, simulation still runs, but only displays signal information on the schematic or layout.

You can choose where you want the waveform window to be displayed. The default, "Cascade", requests that it appears in a new overlapping window. You can also request that the waveform window "Tile Horizontally" or "Tile Vertically", which causes the simulated circuit's window to split, showing the waveform in the other half.

For signals that are aggregated into busses, you can choose the number base for display of values.

The ALS simulator handles only a fixed number of events before it stops simulating (initially 10,000). If your circuit or analysis is too long, you will have to increase the number of events.

The IRSIM simulator only simulates for the range of time that is displayed. If you zoom out, you may have to modify the test vectors to force it to resimulate for the extra displayed time.

The IRSIM simulator can handle three levels of parasitic information: "Quick" makes quick estimates of area and perimeter; "Local" analyzes each level of hierarchy fully but separately; "Full" analyzes the entire



hierarchy, and can take a long time. In addition, you can specify the ".prm" parameter file that you want IRSIM to use.

Besides the options offered by the dialog, there are some additional commands that can be typed to the waveform window. Use "t" (ALS only) to trace the simulation for the selected signal (and "ft" to dump a full trace of all signals). Use "i" to display information about the selected signal. Use "p" to preserve a snapshot of the simulation window in the database (a cell with the "simulation-snapshot" view is created with artwork components). To stop simulation, close the simulation window.

Here is a summary of the single-key commands available in digital simulation windows:

Key	Meaning
l	Set selected signal low at main cursor (gate strength)
wl	Set selected signal low at main cursor (node strength, weak)
sl	Set selected signal low at main cursor (VDD/GND strength, strong)
h	Set selected signal high at main cursor (gate strength)
wh	Set selected signal high at main cursor (node strength, weak)
sh	Set selected signal high at main cursor (VDD/GND strength, strong)
x	Set selected signal undefined at main cursor (gate strength)
wx	Set selected signal undefined at main cursor (node strength, weak)
sx	Set selected signal undefined at main cursor (VDD/GND strength, strong)
v	Set a value on the selected bus signal at main cursor (gate strength)
wv	Set a value on the selected bus signal at main cursor (node strength, weak)
sv	Set a value on the selected bus signal at main cursor (VDD/GND strength, strong)
c	Specify a clock on selected signal
e	Delete all vectors on selected signal
i	Print information about selected signal
t	Trace simulation for selected signal (ALS only)
ft	Trace simulation for all signals (ALS only)
b	Aggregate selected signals into a bus signal
a	Add signal to simulation window
r DEL	Remove selected signal from simulation window
p	Preserve snapshot of simulation window in database
u	Resimulate and redisplay the waveform information
?	Print this help message

These window menu functions apply to the digital simulation windows:

- **Zoom out** (show twice as much time)
- **Zoom in** (show half as much time)
- **Special Zoom / Focus on Highlighted** (show from main to extension cursors)
- **Left** (show earlier time)
- **Right** (show later time)



- **Center**, subcommand **Cursor** (shift the time to the locations of the time cursors)
-

 Previous

 Table of
Contents

Next 



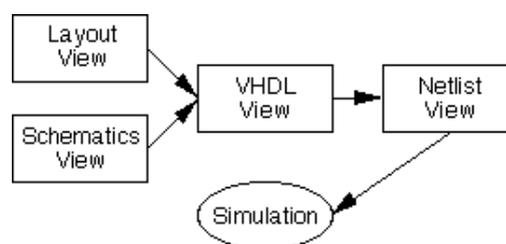
Chapter 10: SIMULATION



10-3: VHDL Interface (ALS)



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 "netlist-als-format". Use the **Edit VHDL View** of the **View** menu to see the VHDL code.



Because cells are kept in memory, it is possible that the size of the VHDL (or more likely the size of the netlist) will be too large. Electric can be instructed to keep either document on disk rather than in a cell, which saves memory. Use the **VHDL Options...** subcommand of the **VHDL Compiler** command of the **Tools** menu and uncheck the "VHDL stored in cell" or "Netlist stored in cell" items to place either or both on disk. See [Section 9-10](#) for more information on the VHDL Compiler.

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, or it can be read from disk using the **Read Text Cell...** command of the **Cells** menu. Also, you can explicitly request that VHDL be produced from schematics or layout with the **Make VHDL View** command of the **View** menu.

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 9-9](#) for more on the Silicon Compiler.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 10: SIMULATION



10-4: Behavioral Models (ALS)



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.

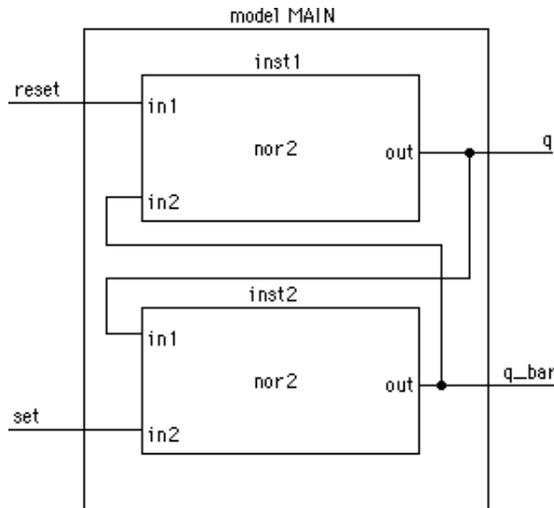
To create (or redefine) a primitive's behavior, simply create the "netlist" view of the cell with that primitive's name. Use the **Edit Cell...** command of the **Cells** menu and select the "netlist-als-format" view. For example, to define the behavior of an ALU cell, edit "alu{net-als}", and to redefine 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.

A library that contains only behavioral models can be built and kept separately from the current library. To identify that library as the location of behavioral models, use the **Select Behavioral Library...** subcommand of the **VHDL Compiler** command of the **Tools** menu.

The netlist format provides three different types of defining 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 C-coded behavior (which must be compiled into Electric, see the module "simalsuser.c"). 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:



The model declaration for the figure is as follows:

```
model main(set, reset, q, q_bar)
inst1: nor2(reset, q_bar, q)
inst2: nor2(q, set, q_bar)
```

The gate description of the nor2 is as follows:

```
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.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 10: SIMULATION

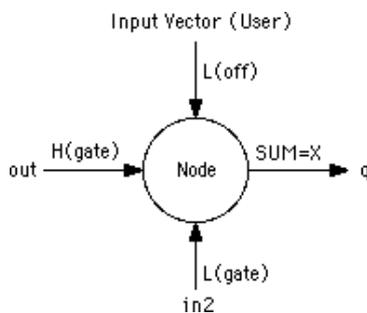


10-5: Simulation Concepts (ALS)



The internal simulation database is composed 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.

The number of events in a simulation is limited to 10,000 events. If you have a more complex simulation that demands more events, change the Maximum Events field in the **Simulation Options...** subcommand of the **Simulation (Built-in)** command of the **Tools** menu.



 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 10: SIMULATION



10-6: The Gate Entity (ALS)



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: – decrement signal by value of operand
= 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

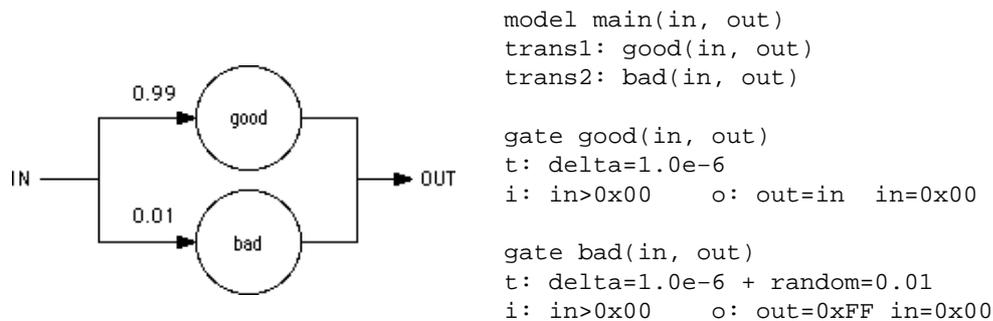
The random probability function enables the user to model things which occur on a percentage basis (e.g. bit error rate, packet routing). The format for random probability declaration is shown below:

Format: *random = value*
Example: *random = 0.75*
 random = 0.25

The value associated with random declaration must be in the range $0.0 \leq \text{value} \leq 1.0$. 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:



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.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 10: SIMULATION



10–7: The Function Entity (ALS)



The function entity is an alternate method of specifying behavior. It makes reference to a C procedure that has been compiled into Electric. Because there are only a limited number of these procedures, 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.

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 C procedure, 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 C procedure does not have to use the values specified in these statements and can schedule events with values that are specified directly inside the routine.

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 C procedure 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.

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 10: SIMULATION



10–8: The Model Entity (ALS)



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).

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 10: SIMULATION



10–9: Documenting the Netlist (ALS)



The designer can document a netlist by inserting comments that begin with the # character. The parser will ignore the rest of the input line when it encounters this symbol. An example illustrating the use of comments is shown below:

```
# Circuit: RS Latch #
#### Model description for the RS latch ####
model main(set, reset, q, q_bar)  inst1: nor2(reset, q_bar, q)
inst2: nor2(q, set, q_bar)
#### Description for 2 input NOR gate ####
gate nor2(in1,in2,out)
t: delta=4.5e-9 + linear=5.0e-10 # Timing Parameters
i: in1=L in2=L  o: out=H@2
i: in1=H       o: out=L@2      # Truth Table Entries
i: in2=H       o: out=L@2
i:             o: out=X@2
load in1=1.5 in2=1.5           # Loading Declaration
```

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)



Chapter 11: INTERPRETERS



11-1: Introduction to Interpreters

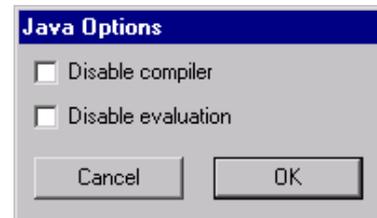


Electric has built in interpretive languages that provide great power in design. The TCL interpreter is a graphical user-interface language. The Lisp interpreter implements a Scheme Lisp dialect. The Java interpreter connects with the Java Virtual Machine.

Note, however, that because of copyright restrictions, these interpreters are not part of the standard GNU distribution and must be obtained separately. The Lisp source code are available from [Static Free Software](#). The Java interpreter is available from [Sun](#). The TCL interpreter is available from [ActiveState](#). For more details, see the installation instructions for UNIX ([Section 1-3](#)), Macintosh ([Section 1-4](#)), and Windows ([Section 1-5](#)).

The **Interpretive Language** command of the **Windows** menu shows the language choices available: **TCL...**, **LISP...** and/or **Java...** Once these commands are issued, you are in direct communication with the interpreter and may type arbitrary expressions in the messages window. When done with an interpretive session, type Ctrl-D to return to Electric (hold the control key and type "D"). On Windows systems, type ESC.

The Java interpreter has two options, invoked with the **Java Options...** subcommand of the **Interpretive Language** command of the **Windows** menu. You can choose to disable the Java compiler, forcing interpretation at each step (useful when debugging). You can also choose to disable Java evaluation completely, causing all expressions to appear in their "source" form (this is useful when documenting code: the expressions will appear in the place of their evaluated values).



Besides basic expressions in the language, it is possible to examine and modify the Electric database. Special language extensions exist for doing this. For more information, see the following sections on Lisp ([Section 11-2](#)), TCL ([Section 11-3](#)), and Java ([Section 11-4](#)).

Another way to make use of the interpretive languages is to place code on nonlayout text, cell parameters, and attributes on Electric objects. In the **Define** subcommand of the **Attributes...** command of the **Info** menu, and in the **Get Info** command when text is selected, Change the "Not CODE" popup entry to "TCL", "LISP", or "JAVA" and type the code in the "Value" field.

[Previous](#)

[Table of Contents](#)

[Next](#)



Chapter 11: INTERPRETERS



11–2: The Lisp Interface



This section explains the Lisp interpretive language interface in the Electric VLSI design system. This interface is built on top of ELK Lisp 3.0, a Scheme dialect (see <http://www-rn.informatik.uni-bremen.de/software/elk/elk.html>).

Throughout this section, examples of Lisp code will appear underlined. For example, the "getarcproto" predicate takes the name of an arc prototype and returns a pointer to that object. This is coded as (getarcproto 'Metal-1) which evaluates to the pointer of the form #[arcproto Metal-1].

This section assumes that the reader is very familiar with the use of Electric, and somewhat familiar with the internals of the system. The Internals Manual (a document that is available from [Static Free Software](#)) provides a broad, C-oriented view of the information described here. For users of Lisp, however, this section summarizes the relevant aspects of the Internals Manual. In general, the best way to understand this section is to try each command as it is explained.

Session Control

To invoke the Lisp interpreter, use the **LISP...** subcommand of the **Language Interpreter** command of the **Tools** menu. On some systems it may be necessary to move the cursor into the messages window (the text window) in order for the interpreter to "hear" you.

If you have a disk file with Lisp code in it, you can read it into the interpreter by typing:
(load 'FILENAME)

To get back to Electric from Lisp, type ^D (hold the Control key and type a "D"). On Windows, you must type the ESC key instead.

Database Structure

The entire Electric database is a collection of objects, each of which has an arbitrary number of attributes. This section briefly outlines the object types and shows how they are related. Further detail can be found in the Internals Manual. See [Section 11–5](#) for a list of attributes on these objects.

Individual components inside of circuits are described with NODEINST objects (instances of nodes), and individual wires are described with ARCINST objects (instances of arcs). Connections between components and wires are described with PORTARCINST objects (instances of ports that connect to arcs). Because both components and wires have geometry, each one also has an associated GEOM object, and all of the GEOM objects in a cell are organized spatially into an R-tree with a collection of RTNODE objects.



Class objects also exist to describe all individuals of a given type. The NODEPROTO object describes the prototypical component, which may have many individual NODEINST objects associated with it. For example, the CMOS P-Transistor is described with a single NODEPROTO object, and many NODEINST objects for each instance of such a transistor in any circuit. Hierarchy is implemented by having complex components, better known as cells, represented in the same way as the primitive components such as transistors. For example, the ALU circuit is described with a single NODEPROTO object, and each instance of that circuit higher up the hierarchy is described with a NODEINST object.

The CELL object aggregates different views and versions of a circuit. Each of these is called a "cell" (represented with a NODEPROTO object) and a cell has both a VIEW pointer and a version number.

In addition to component prototypes, the ARCPROTO describes classes of wires and the PORTPROTO describes classes of component-wire connections. An additional object, the PORTEXPINST, exists for exports. The NETWORK object describes electrically connected ARCINST and PORTPROTO objects within a CELL.

As a further aggregation of objects, the LIBRARY is a collection of cells and cells. The TECHNOLOGY is a collection of primitive components (NODEPROTOS) and all wire classes (ARCPROTOS).

In addition to the above object pointers, there are some standard types of values that can be accessed through getval:

<u>integer</u>	32-bit integer
<u>address</u>	32-bit integer
<u>char</u>	8-bit byte
<u>string</u>	null-terminated string of bytes
<u>float</u>	32-bit floating point number
<u>double</u>	64-bit floating point number
<u>fract</u>	fractional integer number that is multiplied by 120
<u>short</u>	16-bit integer
<u>window</u>	window partition object
<u>window-frame</u>	display window object
<u>constraint</u>	constraint system object
<u>graphics</u>	graphical attributes object

Also, there is the ability to have displayable variables (those whose values appear on the object) with the keyword: displayable.

Database Examination

To begin a search through the database, it is important to know the current library. This is done with: (curlib)

which returns a pointer to a LIBRARY object (for example #[library_noname]). From here, the current cell can be obtained with:

(getval (curlib) 'firstnodeproto)



Essentially, any attribute can be examined with getval, and new attributes can be created with setval.

Getval has the following format:

(getval OBJECT 'ATTRIBUTE)

where OBJECT is the object being accessed and ATTRIBUTE is the attribute being requested. A list of all existing attributes on the Electric objects is given at the end of this document.

New attributes can be created on any object with setval. In general, many of the existing attributes that are described at the end of this document cannot be set with setval, but rather are controlled with special database modification predicates. The format for setval is:

(setval OBJECT 'ATTRIBUTE VALUE OPTIONS)

Where the OPTIONS are either 0 or 'displayable to show this attribute when displaying the object. For example, to add a new attribute called "power-consumption" to the transistor component "t1", and give it the value 75, use:

(setval t1 'power-consumption 75 0)

To add a displayed name to node "t1", use:

(setval t1 'NODE_name "Q1" 'displayable)

To set an array of values, use vectors. For example, to set the shape of pure-layer node "metal" to be a diamond, use:

(setval metal 'trace (vector -1000 0 0 1000 1000 0 0 -1000) 0)

Single entries in array attributes can be set, with:

(setind OBJECT 'ATTRIBUTE INDEX VALUE)

where INDEX is the 0-based entry in the array.

Finally, attributes can be deleted with:

(delval OBJECT 'ATTRIBUTE)

However, only those attributes that have been created with setval can be deleted in this way. The other attributes are protected.

Basic Synthesis

To create a new cell in the current library, use:

(newnodeproto 'CELLNAME (curlib))

which returns a NODEPROTO pointer that can be used in subsequent calls which place components and wires in that cell.

To get the address of an existing NODEPROTO, use:

(getnodeproto 'CELLNAME)

which returns the same type of value as newnodeproto. Thus, the code:

(define mycell (newnodeproto 'adder{lay} (curlib)))

is the same as the code:

(newnodeproto 'adder{lay} (curlib))

(define mycell (getnodeproto 'adder{lay}))

and both deal with the "layout" view of the cell called "adder".

To create a component in a cell, use:

(newnodeinst PROTO LOWX HIGHX LOWY HIGHY TRANSPOSE ANGLE CELL)

where PROTO is a NODEPROTO of the component that is to be created, LOWX, HIGHX, LOWY, and HIGHY are the bounds of the component, ANGLE is the number of tenth-degrees of rotation for the component, TRANSPOSE is nonzero to transpose the component's orientation (after rotation), and CELL is the NODEPROTO in which to place the component.



The four bounds values are somewhat confusing to compute. For primitive components (such as Transistors), any value is acceptable and the component will scale. However, it is still nice to know the default value, which can be obtained from the NODEPROTO with getval as follows:

```
(define tran (getnodeproto 'P-Transistor))  
(define lowx (getval tran 'lowx))  
(define highx (getval tran 'highx))  
(define lowy (getval tran 'lowy))  
(define highy (getval tran 'highy))
```

When complex components (cells) are placed, the bounds **MUST** be exactly the same as the bounding box of the cell's contents. This information is available in the above manner. As an example of newnodeinst, and given the above bounds calculations, a default size P-Transistor is created in cell "adder" with:

```
(define t1 (newnodeinst tran lowx highx lowy highy 0 0 mycell))
```

The returned pointer to the transistor component will be used later when wiring.

To wire two components, it is necessary to know these four things:

- The component objects on the two ends, returned by newnodeinst
- The PORTPROTO values of the connection sites on the components
- The X and Y coordinates of the connection sites
- The type, width, and other characteristics of the wire being created

Connection sites are called PORTPROTOS and are associated with NODEPROTOS. To get the address, use:

```
(getportproto NODEPROTO 'PORTNAME)
```

For example, to get the polysilicon port on the left side of the MOSIS CMOS P-Transistor, use:

```
(define polyleft (getportproto tran 'p-trans-poly-left))
```

Unfortunately, there is no good way to get a list of port names on the primitive components. There are, however, some simplifications. For example, if there is only one port (as is the case with most contacts and pins) then its name is not necessary:

```
(define port (getval tran 'firstportproto))
```

This will obtain the first port on the P-Transistor component. To obtain the coordinates of a port for wiring, use:

```
(portposition NODE PORT)
```

This returns a vector with the coordinates. For example:

```
(define portpos (portposition t1 polyleft))
```

will obtain the coordinate of the "p-trans-poly-left" port on the newly created P-Transistor, t1. The X value will be (vector-ref portpos 0) and the Y value will be (vector-ref portpos 1).

The final piece of information necessary is the type of arc and the width of the arc. Given an arc name, the type can be obtained with:

```
(getarcproto 'ARCNAME)
```

Given an ARCPROTO, its default width can be obtained with:

```
(getval ARCTYPE 'nominalwidth)
```

When all of the information is ready, the call:

```
(newarcinst ARCTYPE WIDTH BITS NODEA PORTA XA YA NODEB PORTB XB YB CELL)
```

places the wire. You can ignore the value of BITS and set it to zero.

Here is a complete example of placing a transistor, a contact, and running a wire between them (the result is shown here).



```

; create a cell called "tran-contact" in the
current library
(define mycell (newnodeproto 'tran-contact
(curlib)))

; get pointers to primitives
(define tran (getnodeproto 'P-Transistor))
(define contact (getnodeproto
'Metal-1-Polysilicon-1-Con))

; get default sizes of these primitives
(define tlowx (getval tran 'lowx))
(define thighx (getval tran 'highx))
(define tlowy (getval tran 'lowy))
(define thighy (getval tran 'highy))
(define clowx (getval contact 'lowx))
(define chighx (getval contact 'highx))
(define clowy (getval contact 'lowy))
(define chighy (getval contact 'highy))

; get pointer to Polysilicon arc and its default
width
(define arctype (getarcproto 'Polysilicon-1))
(define width (getval arctype 'nominalwidth))

; create the transistor and the contact to its left
(define c1 (newnodeinst contact clowx
chighx clowy chighy
0 0 mycell))
(define t1 (newnodeinst tran (+ tlowx 8000)
(+ thighx 8000)
tlowy thighy 0 0 mycell))

; get the transistor's left port coordinates
(define tport (getportproto tran
'p-trans-poly-left))
(define tpos (portposition t1 tport))

; get the contacts's only port coordinates
(define cport (getval contact 'firstportproto))
(define cpos (portposition c1 cport))

; run a wire between the primitives
(newarcinst arctype width 0
t1 tport (vector-ref tpos 0) (vector-ref
tpos 1)
c1 cport (vector-ref cpos 0) (vector-ref
cpos 1) mycell)

```



Hierarchy

Cells, as created by newnodeproto, can be placed in other cells with newnodeinst. The instances simply use complex NODEPROTO fields rather than primitive NODEPROTOs as in the above example. For example, the following code creates a new cell called "two-trans" and places two instances of the above "tran-contact" cell, one above the other.

```

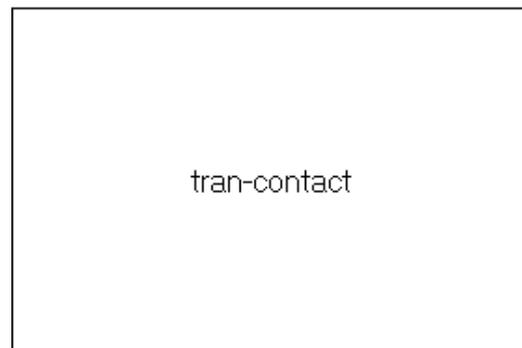
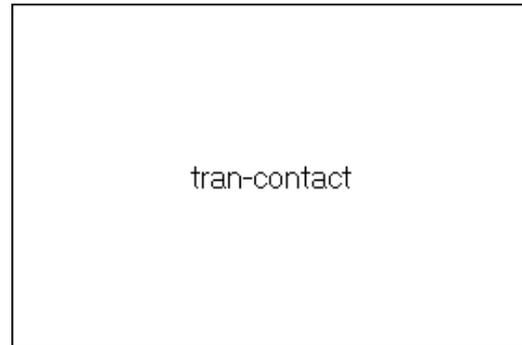
; create a cell called "two-trans"
(define highercell (newnodeproto
'two-trans (curlib)))

; get pointer to the "tran-contact"
cell
(define t-c (getnodeproto
'tran-contact))

; get size of this cell
(define lowx (getval t-c 'lowx))
(define highx (getval t-c 'highx))
(define lowy (getval t-c 'lowy))
(define highy (getval t-c 'highy))

; create the two cell instances,
one above the other
(define o1 (newnodeinst t-c lowx
highx lowy highy
0 0 highercell))
(define o2 (newnodeinst t-c lowx
highx
(+ lowy 10000) (+ highy
10000) 0 0 highercell))

```



Another necessary feature, when making hierarchy, is the ability to place wires between connection sites on cell instances. To do this, it is necessary to create exports. This takes a port on a primitive component (for example, the transistor or contact in the "tran-contact" cell) and makes it into an export on the current cell. This is done with:

```
(newportproto CELL NODE-IN-CELL PORT-ON-NODE 'PORTNAME)
```

where CELL is the cell containing the component whose port is being exported, NODE-IN-CELL is that component, and PORT-ON-NODE is the particular port on that node being exported. For example, to export the top and bottom diffusion ports in the "tran-contact" cell (as shown here), the following code can be added:

```

(newportproto mycell t1
(getportproto tran
'p-trans-diff-top) 'topdiff)
(newportproto mycell t1
(getportproto tran
'p-trans-diff-bottom) 'botdiff)

```



And then, the components "o1" and "o2" in the cell "two-trans" can be wired, using the ports called "topdiff" and "botdiff":

```

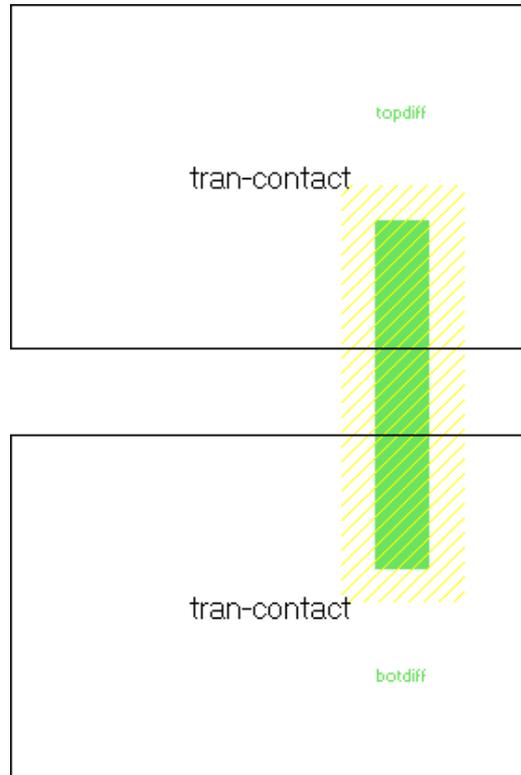
; get pointer to P-Active arc and its
default width
(define darctype (getarcproto
P-Active))
(define dwidth (getval darctype
'nominalwidth))

; get the bottom cell's top port
(define lowport (getportproto mycell
'topdiff))
(define lowpos (portposition o1
lowport))

; get the top cell's bottom port
(define highport (getportproto
mycell 'botdiff))
(define highpos (portposition o2
highport))

; run a wire between the primitives
(newarcinst darctype dwidth 0
o1 lowport (vector-ref lowpos 0)
(vector-ref lowpos 1)
o2 highport (vector-ref highpos
0)
(vector-ref highpos 1)
highercell)

```



Modification

Two types of modification can be done to existing objects: deletion and change. To delete a cell, use: (killnodeproto CELL)

To make a copy of a cell (within the same library or from one library to another), use:

```
(copynodeproto FROM-CELL TO-LIBRARY "TO-CELL-NAME")
```

where FROM-CELL is the original cell (NODEPROTO) and TO-LIBRARY is the destination library. Use (curlib) to copy to the same library. The new cell name is the last parameter. The predicate returns the address of the new cell (NODEPROTO).

To delete a component, use:

```
(killnodeinst NODE)
```

Before a component can be deleted, all wires and exports must be removed.

To change the size or orientation of a component, use:

```
(modifynodeinst NODE DLOWX DLOWY DHIGHX DHIGHY DROTATION DTRN)
```

where DLOWX, DLOWY, DHIGHX, and DHIGHY are the changes to position and size. DROTATION and DTRN are changes to the orientation.



To change the prototype of a component, use:

(replacenodeinst OLD-NODE NEW-PROTOTYPE)

where the old component is OLD-NODE, and the new NODEPROTO that should be in its place is NEW-PROTOTYPE. This new prototype must be able to connect to all existing arcs. The predicate returns the address of the new component.

To delete a wire, use:

(killarcinst ARC)

To change the width or position of a wire, use:

(modifyarcinst ARC DWIDTH DX1 DY1 DX2 DY2)

where DWIDTH, DX1, DY1, DX2, and DY2 are the changes to the width, X/Y position of end 1, and X/Y position of end 2. Note that position changes cannot cause the connecting nodes to move, so the changes may only be small ones that work within the ports.

To change the prototype of a wire, use:

(replacearcinst OLD-ARC NEW-PROTOTYPE)

where OLD-ARC is the former wire and NEW-PROTOTYPE is the new ARCPROTO to use. The nodes on either end must be able to accept this new type of wire. The predicate returns the address of the new wire.

To delete an export, use:

(killportproto CELL PORT)

which will remove port PORT on cell CELL.

To move an export from one component to another (keeping connected wires), use:

(moveportproto CELL OLD-PORT NEW-NODE PORT-ON-NEW-NODE)

where the old port is OLD-PORT in cell CELL, and it is now moved to component NEW-NODE (which is also in cell CELL), port PORT-ON-NEW-NODE of that component.

Search

A common operation is a search of all components in a cell. The following code prints the name of all components in the cell "mycell":

```
(do_
  (
    (node (getval mycell 'firstnodeinst)____
      (getval node 'nextnodeinst))____
    )
  ((null? node))____

  (format #t "Found ~s node~%" (describenode node))____
) _
```

Where describenode is defined as follows (the name of a node is found in different places depending on whether it is a primitive or complex NODEPROTO):

```
(define describenode_
  (lambda (node)____
    (define proto (getval node 'proto))____
    (if (= (getval proto 'primindex) 0)____
        (getval (getval proto 'cell) 'cellname)____
```



```

    (getval proto 'primname)_____
  ) -
) -
) -

```

And the following code prints the name of all wires in the cell "mycell":

```

(do_
  (
    -
    (arc (getval mycell 'firstarcinst)_____
      (getval arc 'nextarcinst))_____
    ) -
    ((null? arc))_____

    (format #t "Found ~s arc~%"
      (getval (getval arc 'proto) 'protoname))_____
  ) -

```

To do a search of all nodes and arcs in a rectangular area of a cell, first call:

```
(initsearch LOWX HIGHX LOWY HIGHY CELL)
```

where LOWX, HIGHX, LOWY, and HIGHY are the coordinates to search in cell CELL (a NODEPROTO).

This predicate will return a search key that can then be passed repeatedly to:

```
(nextobject SEARCHKEY)
```

which will return GEOM objects of each node and arc in the search area. When this predicate returns Null, the search is complete. GEOM objects can point to either nodes or arcs, depending on their "entryisnode" attribute. Then, the "entryaddr" attribute will point to the actual NODEINST or ARCINST. Here is an example of code that prints the names of all nodes and arcs in the area (2000

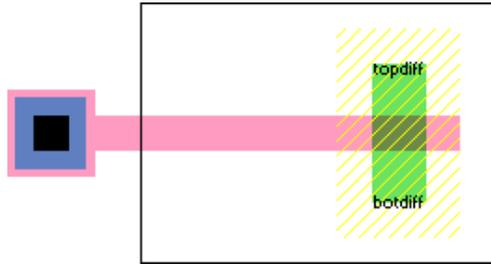
```

(define key (initsearch 2000 10000 -3000 3000 mycell))
(do_
  (
    -
    (object (nextobject key) (nextobject key))_____
  ) -
  ((null? object))_____

  (define isnode (getval object 'entryisnode))_____
  (if (= isnode 0)_____
    (format t "Found ~s arc~%"
      (getval_____
        (getval _____
          (getval object 'entryaddr)_____
          'proto) _____
          'protoname)_____
        ) -
      (format t "Found ~s node~%"
        (describenode (getval object 'entryaddr))_____
        ) -
      ) -
  ) -

```





Views

A view is an object that describes a cell. There are many standard views: Layout, Schematic, Icon, Simulation–snapshot, Skeleton, VHDL, Verilog, Document, Unknown, and many flavors of Netlist. In addition, new views can be created with "newview":

(newview 'VIEWNAME 'ABBREVIATION)

and views can be deleted with killview (the standard views cannot be deleted):

(killview VIEW)

To get a view object, use getview on its name.

To associate different views of a cell, the predicates iconview and contentsview obtain different cells. For example:

(iconview mycell)

finds the associated icon cell of the cell in which "mycell" resides.

Libraries

In the above examples, the current library was always used. This is determined by calling:

(curlib)

However, there can be other libraries. To get a specific named library, use:

(getlibrary 'LIBNAME)

To create a new library, use:

(newlibrary 'LIBRARYNAME 'LIBRARYFILE)

where LIBRARYNAME is the name to use, and LIBRARYFILE is the path name where this library will be saved. This predicate returns the address of a new library object that can then be used when creating cells.

Only one library is the current one, and to switch, you must use:

(selectlibrary LIBRARY)

A library can be deleted with:

(killlibrary LIBRARY)

A library can be erased (its cells deleted, but not the library) with:

(eraselibrary LIBRARY)

Technologies

A technology is an environment of design that includes primitive components and wire prototypes. The current technology can be obtained with:

(curtech)



A specific technology can be obtained from its name with:

(gettechnology 'TECHNAME')

All technologies can be found by traversing a linked list, the head of which is a technology named "Generic".

Tools

A tool is a piece of synthesis or analysis code that can operate upon the database. A particular tool object can be obtained with:

(gettool 'TOOLNAME')

where the possible names of tools are:

<u>compaction</u>	circuit compaction
<u>compensation</u>	geometry compensation
<u>drc</u>	design-rule checking
<u>erc</u>	electrical-rule checking
<u>io</u>	input/output control
<u>logeffort</u>	logical effort analysis
<u>network</u>	network maintenance
<u>pla</u>	programmable logic array generator
<u>project</u>	project management
<u>routing</u>	automatic wire routing
<u>silicon-compiler</u>	netlist-to-layout silicon assembler
<u>simulation</u>	simulation
<u>user</u>	the user interface
<u>vhdl-compiler</u>	VHDL-to-netlist compiler

The number of tools is available with:

(maxtool)

And a particular tool, indexed from 0 to (maxtool)-1 can be obtained with:

(indextool INDEX)

A tool can be switched on with:

(toolturion TOOL)

where TOOL is a tool object.

A tool can be turned off with:

(toolturionoff TOOL)

A tool can be given a specific instruction with:

(tellttool TOOL PARAMETERS)

For example, to list all technologies, use this code:

(tellttool (gettool 'user') 'show 'technologies)



These commands are from the low-level command interpreter, which is documented fully in the Internals Manual.

Miscellaneous

Every change to the database is queued internally in a "batch" which includes the change and any constrained side-effects of that change. A new batch is created for each Lisp session with the interpreter (also for each Electric command that is issued from the keyboard/mouse). To reverse the last batch of changes, use:

(undoabatch)

Multiple calls to this predicate in a single batch will undo multiple batches. To erase the list of change batches, use:

(nundoallowed)

If you are creating a wire that makes many bends, it is necessary to create special nodes called "pins" at each bend. To find out what kind of pin to use for a given wire type, use:

(getpinproto ARC-PROTO)

where ARC-PROTO is the wire type, and the predicate returns the component type (NODEPROTO) of the pin.

Network objects can be obtained by name with the predicate getnetwork which takes a name and a cell in which to search. For example, the code:

(getnetwork 'insig mycell)

obtains the address of the network called "insig" in cell mycell.

The generic function of a node instance can be determined with:

(nodefunction NODE)

which returns a value from the list of constants in the C header file "efunction.h". This value is essentially the same one as would be obtained by looking at the "userbits" field of the node's prototype. However, certain components that have generic prototypes will be made more specific by this predicate.

To get an attribute value from an instance above this in the hierarchy, use:

(getparentval 'name DEFAULT HEIGHT)

where name is the attribute name, DEFAULT is the default value to return if the attribute is not found, and HEIGHT is the number of levels of hierarchy to climb when looking for the attribute (0 for infinite). As a shortcut for finding parameter values, there are four macros which use this routine:

- (P 'xx) obtains the value of parameter xx from the parent instance in the hierarchy.
- (PD 'xx def) obtains the value of parameter xx from the parent instance in the hierarchy and uses the value def if the parameter cannot be found.
- (PAR 'xx) obtains the value of parameter xx from any higher instance, anywhere in the hierarchy.
- (PARD 'xx def) obtains the value of parameter xx from any higher instance, anywhere in the hierarchy and uses the value def if the parameter cannot be found.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 11: INTERPRETERS



11-3: The TCL Interface



This section explains the TCL interpretive language interface in the Electric VLSI design system.

Throughout this section, examples of TCL code will appear underlined. For example, the "getarcproto" function takes the name of an arc prototype and returns a pointer to that object. This is coded as getarcproto Metal-1 which evaluates to the pointer of the form #arcproto15391808.

This section assumes that the reader is very familiar with the use of Electric, and somewhat familiar with the internals of the system. The Internals Manual (a document that is available from [Static Free Software](#)) provides a broad, C-oriented view of the information described here. For users of TCL, however, this section summarizes the relevant aspects of the Internals Manual. In general, the best way to understand this section is to try each command as it is explained.

Session Control

To invoke the TCL interpreter, use the **TCL...** subcommand of the **Language Interpreter** command of the **Tools** menu. On some systems it may be necessary to move the cursor into the messages window (the text window) in order for the interpreter to "hear" you.

If you have a disk file with TCL code in it, you can read it into the interpreter by typing:
source FILENAME

To get back to Electric from TCL, type ^D (hold the Control key and type a "D"). On Windows, you must type the ESC key instead.

Database Structure

The entire Electric database is a collection of objects, each of which has an arbitrary number of attributes. This section briefly outlines the object types and shows how they are related. Further detail can be found in the Internals Manual. See [Section 11-5](#) for a list of attributes on these objects.

Individual components inside of circuits are described with nodeinst objects (instances of nodes), and individual wires are described with arcinst objects (instances of arcs). Connections between components and wires are described with portarcinst objects (instances of ports that connect to arcs). Because both components and wires have geometry, each one also has an associated geom object, and all of the geom objects in a cell are organized spatially into an R-tree with a collection of rnode objects.

Class objects also exist to describe all individuals of a given type. The nodeproto object describes the



prototypical component, which may have many individual nodeinst objects associated with it. For example, the CMOS P-Transistor is described with a single nodeproto object, and many nodeinst objects for each instance of such a transistor in any circuit. Hierarchy is implemented by having complex components, better known as cells, represented in the same way as the primitive components such as transistors. For example, the ALU circuit is described with a single nodeproto object, and each instance of that circuit higher up the hierarchy is described with a nodeinst object.

The cell object aggregates different views and versions of a circuit. Each of these is called a "cell" (represented with a nodeproto object) and a cell has both a view pointer and a version number.

In addition to component prototypes, the arcproto describes classes of wires and the portproto describes classes of component-wire connections. An additional object, the portexpinst, exists for exports. The network object describes electrically connected arcinst and portproto objects within a cell.

As a further aggregation of objects, the library is a collection of cells and cells. The technology is a collection of primitive components (nodeprotos) and all wire classes (arcprotos).

In addition to the above object pointers, there are some standard types of values that can be accessed through getval:

<u>integer</u>	32-bit integer
<u>string</u>	null-terminated string of bytes
<u>float</u>	32-bit floating point number
<u>window</u>	window partition object
<u>windowframe</u>	display window object
<u>constraint</u>	constraint system object
<u>graphics</u>	graphical attributes object

Also, there is the ability to have displayable variables (those whose values appear on the object) with the keyword: displayable.

Database Examination

To begin a search through the database, it is important to know the current library. This is done with:

curlib

which returns a pointer to a library object (for example #library15464800). From here, the current cell can be obtained with:

getval [curlib] firstnodeproto

Essentially, any attribute can be examined with getval, and new attributes can be created with setval.

Getval has the following format:

getval Object Attribute

where Object is the object being accessed and Attribute is the attribute being requested. A list of all existing attributes on the Electric objects is given at the end of this document.



New attributes can be created on any object with setval. In general, many of the existing attributes that are described at the end of this document cannot be set with setval, but rather are controlled with special database modification predicates. The format for setval is:

setval Object Attribute Value Options

Where the Options are either 0 or displayable to show this attribute when displaying the object. For example, to add a new attribute called "power-consumption" to the transistor component "t1", and give it the value 75, use:

setval \$t1 power-consumption 75

To add a displayed name to node "t1", use:

setval \$t1 NODE name 01 displayable

To set an array of values, use lists. For example, to set the shape of pure-layer node "metal" to be a diamond, use:

setval \$metal trace {-1000 0 0 1000 1000 0 0 -1000}

Single entries in array attributes can be set, with:

setind Object Attribute Index Value

where Index is the 0-based entry in the array.

Finally, attributes can be deleted with:

delval Object Attribute

However, only those attributes that have been created with setval can be deleted in this way. The other attributes are protected.

Basic Synthesis

To create a new cell in the current library, use:

newnodeproto CellName [curlib]

which returns a nodeproto pointer that can be used in subsequent calls which place components and wires in that cell.

To get the address of an existing nodeproto, use:

getnodeproto CellName

which returns the same type of value as newnodeproto. Thus, the code:

set mycell [newnodeproto "adder{lay}" [curlib]]

is the same as the code:

newnodeproto "adder{lay}" [curlib]

set mycell [getnodeproto "adder{lay}"]

and both deal with the "layout" view of the cell called "adder".

To create a component in a cell, use:

newnodeinst Proto LowX HighX LowY HighY Transpose Angle Cell

where Proto is a nodeproto of the component that is to be created, LowX, HighX, LowY, and HighY are the bounds of the component, Angle is the number of tenth-degrees of rotation for the component, Transpose is nonzero to transpose the component's orientation (after rotation), and Cell is the nodeproto in which to place the component.

The four bounds values are somewhat confusing to compute. For primitive components (such as Transistors), any value is acceptable and the component will scale. However, it is still nice to know the default value, which can be obtained from the nodeproto with getval as follows:

set tran [getnodeproto P-Transistor]

set lowx [getval \$tran lowx]

set highx [getval \$tran highx]



```
set lowy [getval $stran lowy]
```

```
set highy [getval $stran highy]
```

When complex components (cells) are placed, the bounds MUST be exactly the same as the bounding box of the cell's contents. This information is available in the above manner. As an example of newnodeinst, and given the above bounds calculations, a default size P-Transistor is created in cell "adder" with:

```
set t1 [newnodeinst $stran $lowx $highx $lowy $highy 0 0 $mycell]
```

The returned pointer to the transistor component will be used later when wiring.

To wire two components, it is necessary to know these four things:

- The component objects on the two ends, returned by newnodeinst
- The portproto values of the connection sites on the components
- The X and Y coordinates of the connection sites
- The type, width, and other characteristics of the wire being created

Connection sites are called portprotos and are associated with nodeprotos. To get the address, use:

```
getportproto NodeProto PortName
```

For example, to get the polysilicon port on the left side of the MOSIS CMOS P-Transistor, use:

```
set polyleft [getportproto $stran p-trans-poly-left]
```

Unfortunately, there is no good way to get a list of port names on the primitive components. There are, however, some simplifications. For example, if there is only one port (as is the case with most contacts and pins) then its name is not necessary:

```
set port [getval $stran firstportproto]
```

This will obtain the first port on the P-Transistor component. To obtain the coordinates of a port for wiring, use

```
portposition Node Port
```

This returns a list with the coordinates. For example:

```
set portpos [portposition $t1 $polyleft]
```

will obtain the coordinate of the "p-trans-poly-left" port on the newly created P-Transistor, t1. The X value will be lindex \$portpos 0 and the Y value will be lindex \$portpos 1.

The final piece of information necessary is the type of arc and the width of the arc. Given an arc name, the type can be obtained with:

```
getarcproto ArcName
```

Given an arcproto, its default width can be obtained with:

```
getval Arc nominalwidth
```

When all of the information is ready, the call:

```
newarcinst ArcType Width Bits NodeA PortA XA YA NodeB PortB XB YB Cell
```

places the wire. You can ignore the value of Bits and set it to zero.

Here is a complete example of placing a transistor, a contact, and running a wire between them (the result is shown here).

```
# create a cell called "tran-contact" in the  
current library
```

```
set mycell [newnodeproto tran-contact  
[curlib]]
```

```
# get pointers to primitives
```

```
set tran [getnodeproto P-Transistor]
```

```
set contact [getnodeproto  
Metal-1-Polysilicon-1-Con]
```



```

# get default sizes of these primitives
set flowx [getval $tran lowx]
set thighx [getval $tran highx]
set tlowy [getval $tran lowy]
set thighy [getval $tran highy]
set clowx [getval $contact lowx]
set chighx [getval $contact highx]
set clowy [getval $contact lowy]
set chighy [getval $contact highy]

# get pointer to Polysilicon arc and its
default width
set arctype [getarcproto Polysilicon-1]
set width [getval $arctype nominalwidth]

# create the transistor and the contact to its
left
set c1 [newnodeinst $contact $clowx
$chighx
$clowy $chighy 0 0 $mycell]
set t1 [newnodeinst $tran [expr
$flowx+8000]
[expr $thighx+8000] $tlowy $thighy 0
0 $mycell]

# get the transistor's left port coordinates
set tport [getportproto $tran
p-trans-poly-left]
set tpos [portposition $t1 $tport]

# get the contacts's only port coordinates
set cport [getval $contact firstportproto]
set cpos [portposition $c1 $cport]

# run a wire between the primitives
newarcinst $arctype $width 0
$t1 $tport [lindex $tpos 0] [lindex $tpos
1]
$c1 $cport [lindex $cpos 0] [lindex
$cpos 1] $mycell

```

Hierarchy

Cells, as created by `newnodeproto`, can be placed in other cells with `newnodeinst`. The instances simply use complex `nodeproto` fields rather than primitive `nodeprotos` as in the above example. For example, the following code creates a new cell called "two-trans" and places two instances of the above "tran-contact" cell, one above the other.



```

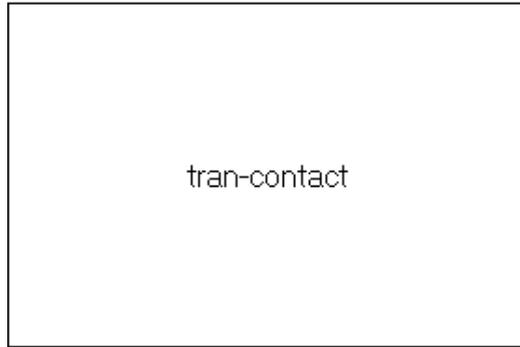
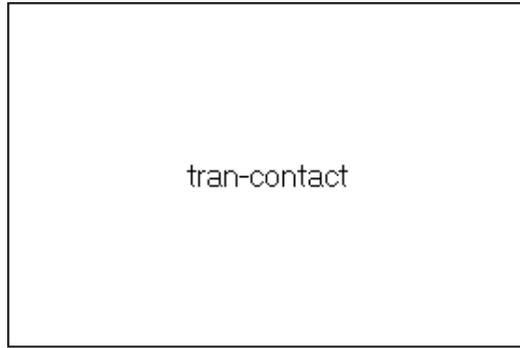
# create a cell called "two-trans"
set highercell [newnodeproto
two-trans [curlib]]

# get pointer to the
"tran-contact" cell
set tc [getnodeproto tran-contact]

# get size of this cell
set lowx [getval $tc lowx]
set highx [getval $tc highx]
set lowy [getval $tc lowy]
set highy [getval $tc highy]

# create the two cell instances,
one above the other
set o1 [newnodeinst $tc $lowx
$highx $lowy $highy
0 0 $highercell]
set o2 [newnodeinst $tc $lowx
$highx
[expr $lowy+10000] [expr
$highy+10000]
0 0 $highercell]

```



Another necessary feature, when making hierarchy, is the ability to place wires between connection sites on cell instances. To do this, it is necessary to create exports. This takes a port on a primitive component (for example, the transistor or contact in the "tran-contact" cell) and makes it into an export on the current cell. This is done with:

```
newportproto Cell NodeInCell PortOnNode PortName
```

where Cell is the cell containing the component whose port is being exported, NodeInCell is that component, and PortOnNode is the particular port on that node being exported. For example, to export the top and bottom diffusion ports in the "tran-contact" cell (as shown here), the following code can be added:

```

newportproto $mycell $t1
[getportproto $tran
p-trans-diff-top] topdiff
newportproto $mycell $t1
[getportproto $tran
p-trans-diff-bottom] botdiff

```



And then, the components "o1" and "o2" in the cell "two-trans" can be wired, using the ports called "topdiff" and "botdiff":



```

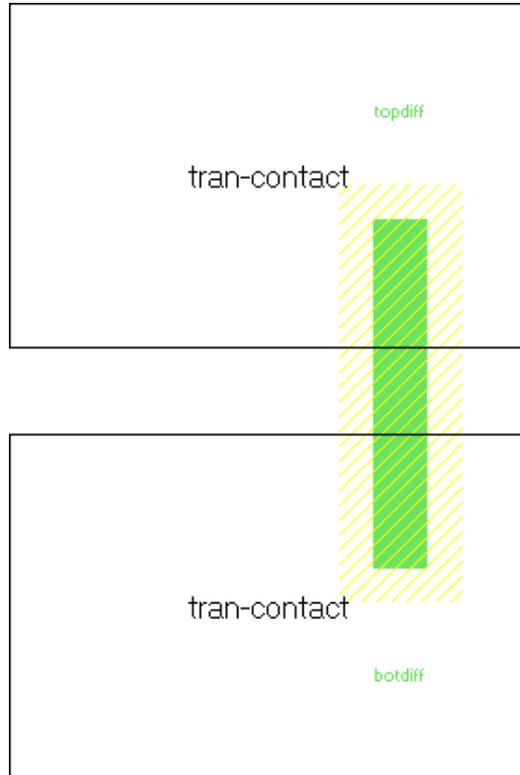
# get pointer to P-Active arc and
its default width
set darctype [getarcproto
P-Active]
set dwidth [getval $darctype
nominalwidth]

# get the bottom cell's top port
set lowport [getportproto $mycell
topdiff]
set lowpos [portposition $o1
$lowport]

# get the top cell's bottom port
set highport [getportproto $mycell
botdiff]
set highpos [portposition $o2
$highport]

# run a wire between the
primitives
newarcinst $darctype $dwidth 0
$o1 $lowport [lindex $lowpos
0] [lindex $lowpos 1]
$o2 $highport [lindex
$highpos 0]
[lindex $highpos 1]
$highercell

```



Modification

Two types of modification can be done to existing objects: deletion and change. To delete a cell, use:
killnodeproto Cell

To make a copy of a cell (within the same library or from one library to another), use:
copynodeproto FromCell ToLibrary ToCellName
 where FromCell is the original cell (nodeproto) and ToLibrary is the destination library. Use curlib to copy to the same library. The new cell name is the last parameter. The predicate returns the address of the new cell (nodeproto).

To delete a component, use:
killnodeinst Node
 Before a component can be deleted, all wires and exports must be removed.

To change the size or orientation of a component, use:
modifynodeinst Node DLowX DLowY DHighX DHighY DRotation DTrans
 where DLowX, DLowY, DHighX, and DHighY are the changes to position and size. DRotation and DTrans are changes to the orientation.

To change the prototype of a component, use:
replacnodeinst OldNode NewPrototype



where the old component is OldNode, and the new nodeproto that should be in its place is NewPrototype. This new prototype must be able to connect to all existing arcs. The predicate returns the address of the new component.

To delete a wire, use:

killarcinst Arc

To change the width or position of a wire, use:

modifarcinst Arc DWidth DX1 DY1 DX2 DY2

where DWidth, DX1, DY1, DX2, and DY2 are the changes to the width, X/Y position of end 1, and X/Y position of end 2. Note that position changes cannot cause the connecting nodes to move, so the changes may only be small ones that work within the ports.

To change the prototype of a wire, use:

replacearcinst OldArc NewPrototype

where OldArc is the former wire and NewPrototype is the new arcproto to use. The nodes on either end must be able to accept this new type of wire. The predicate returns the address of the new wire.

To delete an export, use:

killportproto Cell Port

which will remove port Port on cell Cell.

To move an export from one component to another (keeping connected wires), use:

moveportproto Cell OldPort NewNode PortOnNewNode

where the old port is OldPort in cell Cell, and it is now moved to component NewNode (which is also in cell Cell), port PortOnNewNode of that component.

Search

A common operation is a search of all components in a cell. The following code prints the name of all components in the cell "mycell":

```
for { set node [getval $mycell firstnodeinst] } _  
{ [string c $node #nodeinst-1] != 0 } _  
{ set node [getval $node nextnodeinst] } _  
{ _  
puts stdout [format "Found %s node" [describenode $node]] _  
} _
```

Where describenode is defined as follows (the name of a node is found in different places depending on whether it is a primitive or complex nodeproto):

```
proc describenode node _  
{ _  
set proto [getval $node proto] _  
if { [getval $proto primindex] == 0 } _  
{ return [getval [getval $proto cell] cellname] }  
return [getval $proto primname] _  
} _
```

And the following code prints the name of all wires in the cell "mycell":



```

for { set arc [getval $mycell firstarcinst] }
{ [string c $arc #arcinst-1] != 0 }
{ set arc [getval $arc nextarcinst] }
{ _
puts stdout [format "Found %s arc"
[getval [getval $arc proto] protoname]]
} _

```

To do a search of all nodes and arcs in a rectangular area of a cell, first call:

```
initsearch LowX HighX LowY HighY Cell
```

where LowX, HighX, LowY, and HighY are the coordinates to search in cell Cell (a nodeproto). This predicate will return a search key that can then be passed repeatedly to:

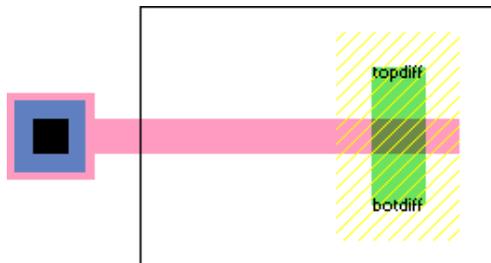
```
nextobject SearchKey
```

which will return geom objects of each node and arc in the search area. When this predicate returns #geom-1, the search is complete. geom objects can point to either nodes or arcs, depending on their "entryisnode" attribute. Then, the "entryaddr" attribute will point to the actual nodeinst or arcinst. Here is an example of code that prints the names of all nodes and arcs in the area (2000

```

set key [initsearch 2000 10000 -3000 3000 $mycell]
for { set object [nextobject $key] }
{ [string c $object #geom-1] != 0 }
{ set object [nextobject $key] }
{ _
set isnode [getval $object entryisnode]
if { $isnode }
{ puts stdout [format "Found %s node"
[describenode [getval $object entryaddr]]] }
else
{ puts stdout [format "Found %s arc" [getval [getval [
getval $object entryaddr] proto] protoname]] }
} _

```



Views

A view is an object that describes a cell. There are many standard views: Layout, Schematic, Icon, Simulation-snapshot, Skeleton, VHDL, Verilog, Document, Unknown, and many flavors of Netlist. In addition, new views can be created with "newview":

```
newview ViewName Abbreviation
```

and views can be deleted with killview (the standard views cannot be deleted):

```
killview View
```

To get a view object, use getview on its name.



To associate different views of a cell, the predicates iconview and contentsview obtain different cells. For example:

iconview Mycell

finds the associated icon cell of the cell in which "Mycell" resides.

Libraries

In the above examples, the current library was always used. This is determined by calling:

curlib

However, there can be other libraries. To get a specific named library, use:

getlibrary LibName

To create a new library, use:

newlibrary LibraryName LibraryFile

where LibraryName is the name to use, and LibraryFile is the path name where this library will be saved.

This predicate returns the address of a new library object that can then be used when creating cells.

Only one library is the current one, and to switch, you must use:

selectlibrary Lib

A library can be deleted with:

killlibrary Lib

A library can be erased (its cells deleted, but not the library) with:

eraselibrary Lib

Technologies

A technology is an environment of design that includes primitive components and wire prototypes. The current technology can be obtained with:

curtech

A specific technology can be obtained from its name with:

gettechnology TechName

All technologies can be found by traversing a linked list, the head of which is a technology named "Generic".

Tools

A tool is a piece of synthesis or analysis code that can operate upon the database. A particular tool object can be obtained with:

gettool ToolName

where the possible names of tools are:

<u>compaction</u>	circuit compaction
<u>compensation</u>	geometry compensation
<u>drc</u>	design-rule checking
<u>erc</u>	electrical-rule checking
<u>io</u>	input/output control



<u>logeffort</u>	logical effort analysis
<u>network</u>	network maintenance
<u>pla</u>	programmable logic array generator
<u>project</u>	project management
<u>routing</u>	automatic wire routing
<u>silicon-compiler</u>	netlist-to-layout silicon assembler
<u>simulation</u>	simulation
<u>user</u>	the user interface
<u>vhdl-compiler</u>	VHDL-to-netlist compiler

The number of tools is available with:

maxtool

And a particular tool, indexed from 0 to (maxtool)-1 can be obtained with:

indextool Index

A tool can be switched on with:

toolturon Tool

where Tool is a tool object.

A tool can be turned off with:

toolturloff Tool

A tool can be given a specific instruction with:

telltool Tool PARAMETERS

For example, to list all technologies, use this code:

telltool [getaid user] {show technologies}

These commands are from the low-level command interpreter, which is documented fully in the Internals Manual.

Miscellaneous

Every change to the database is queued internally in a "batch" which includes the change and any constrained side-effects of that change. A new batch is created for each TCL session with the interpreter (also for each Electric command that is issued from the keyboard/mouse). To reverse the last batch of changes, use:

undoabatch

Multiple calls to this predicate in a single batch will undo multiple batches. To erase the list of change batches, use:

nundoallowed

If you are creating a wire that makes many bends, it is necessary to create special nodes called "pins" at each bend. To find out what kind of pin to use for a given wire type, use:

getpinproto Arc

where Arc is the wire type, and the predicate returns the component type (nodeproto) of the pin.



Network objects can be obtained by name with the predicate getnetwork which takes a name and a cell in which to search. For example, the code:

```
getnetwork insig Mycell
```

obtains the address of the network called "insig" in cell Mycell.

The generic function of a node instance can be determined with:

```
nodefunction Node
```

which returns a value from the list of constants in the C header file "efunction.h". This value is essentially the same one as would be obtained by looking at the "userbits" field of the node's prototype. However, certain components that have generic prototypes will be made more specific by this predicate.

To get an attribute value from an instance above this in the hierarchy, use:

```
getparentval Name Default Height
```

where Name is the attribute name, Default is the default value to return if the attribute is not found, and Height is the number of levels of hierarchy to climb when looking for the attribute (0 for infinite). As a shortcut for finding parameter values, there are four macros which use this routine:

- P_{xx} obtains the value of parameter xx from the parent instance in the hierarchy.
 - PD_{xx} def obtains the value of parameter xx from the parent instance in the hierarchy and uses the value def if the parameter cannot be found.
 - PAR_{xx} obtains the value of parameter xx from any higher instance, anywhere in the hierarchy.
 - PARD_{xx} def obtains the value of parameter xx from any higher instance, anywhere in the hierarchy and uses the value def if the parameter cannot be found.
-

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 11: INTERPRETERS



11-4: The Java Interface



This section explains the Java interpretive language interface in the Electric VLSI design system.

This section assumes that the reader is very familiar with the use of Electric, and somewhat familiar with the internals of the system. The Internals Manual (a document that is available from [Static Free Software](#)) provides a broad, C-oriented view of the information described here. For users of Java, however, this section summarizes the relevant aspects of the Internals Manual. In general, the best way to understand this section is to try each command as it is explained.

Throughout this section, examples of Java code will appear underlined. For example, the "getArcProto" function takes the name of an arc prototype and returns a pointer to that object. This is coded as Electric.getArcProto("Metal-1") which evaluates to the pointer of the form ArcInst(21726672).

Session Control

To invoke the Java interpreter, use the **JAVA...** subcommand of the **Language Interpreter** command of the **Tools** menu. On some systems it may be necessary to move the cursor into the messages window (the text window) in order for the interpreter to "hear" you.

If you have installed the Bean Shell (see the installation instructions) then the Java interpreter will be able to handle any Java expression. The Bean Shell also allows you to use "E" instead of "Electric" in any expression.

If you do not have the Bean Shell installed, then the interpreter can accept only one type of command: CLASS.METHOD which invokes a method in a class. The method must be static and take no parameters.

Electric has its own private area for Java classes, which it adds to the list of places that Java will search. This path is the "java" subdirectory of Electric's library directory. It is best to place your Java code there, and to add this import line at the start:

```
import COM.staticfreesoft.*;
```

To get back to Electric from Java, type ^D (hold the Control key and type a "D"). On Windows, you must type the ESC key instead.

Java used in Parameters

An important use of Java is in attributes and parameters. For example, the "width" attribute of a schematic transistor can be set to be code. Such code takes the form of an expression such as 2+2 or Math.Sqrt(7.5).



Commonly, the expression needs to make use of a parameter value (see [Section 6–8](#) for more on creating cell parameters). To get an parameter value from an instance above this in the hierarchy, use:

`Electric.getParentVal("name", default, height)`

where `"name"` is the attribute name, `default` is the default value to return if the attribute is not found, and `height` is the number of levels of hierarchy to climb when looking for the attribute (0 for infinite). As a shortcut for finding parameter values, the Bean Shell adds five macros which use this routine:

- `P("xx")` obtains the value of parameter `xx` from the parent instance in the hierarchy.
- `@xx` is a shortcut for `P("xx")`.
- `PD("xx", def)` obtains the value of parameter `xx` from the parent instance in the hierarchy and uses the value `def` if the parameter cannot be found.
- `PAR("xx")` obtains the value of parameter `xx` from any higher instance, anywhere in the hierarchy.
- `PARD("xx", def)` obtains the value of parameter `xx` from any higher instance, anywhere in the hierarchy and uses the value `def` if the parameter cannot be found.

So, if the transistor is in a cell with a parameter called "strength", and it should be half that value wide, use the expression `@strength/2`.

Database Structure

The entire Electric database is a collection of objects, each of which has an arbitrary number of attributes. This section briefly outlines the object types and shows how they are related. Further detail can be found in the Internals Manual. See [Section 11–5](#) for a list of attributes on these objects.

Individual components inside of circuits are described with `NodeInst` objects (instances of nodes), and individual wires are described with `ArcInst` objects (instances of arcs). Connections between components and wires are described with `PortArcInst` objects (instances of ports that connect to arcs). Because both components and wires have geometry, each one also has an associated `Geom` object, and all of the `Geom` objects in a cell are organized spatially into an R-tree with a collection of `RTNode` objects.

Class objects also exist to describe all individuals of a given type. The `NodeProto` object describes the prototypical component, which may have many individual `NodeInst` objects associated with it. For example, the CMOS P-Transistor is described with a single `NodeProto` object, and many `NodeInst` objects for each instance of such a transistor in any circuit. Hierarchy is implemented by having complex components, better known as cells, represented in the same way as the primitive components such as transistors. For example, the ALU circuit is described with a single `NodeProto` object, and each instance of that circuit higher up the hierarchy is described with a `NodeInst` object.

The `Cell` object aggregates different views and versions of a circuit. Each of these is called a "cell" (represented with a `NodeProto` object) and a cell has both a `View` pointer and a version number.

In addition to component prototypes, the `ArcProto` describes classes of wires and the `PortProto` describes classes of component-wire connections. An additional object, the `PortExpInst`, exists for exports. The `Network` object describes electrically connected `ArcInst` and `PortProto` objects within a cell.

As a further aggregation of objects, the Library is a collection of cells and cells. The `Technology` is a collection of primitive components (`NodeProtos`) and all wire classes (`ArcProtos`).

In addition to the above object pointers, there are some standard types of values that can be accessed through `getval`:

<code>Integer</code>	32-bit integer
----------------------	----------------



<u>String</u>	null-terminated string of bytes
<u>Float</u>	32-bit floating point number
<u>WindowPart</u>	window partition object
<u>WindowFrame</u>	display window object
<u>Constraint</u>	constraint system object
<u>Graphics</u>	graphical attributes object
<u>Polygon</u>	graphical shape object
<u>XArray</u>	transformation object

Also, there is the ability to have displayable variables (those whose values appear on the object) with the keyword: vdisplay.

Any Java object that represents an Electric object is actually an object with a single field in it: the address of that object in Electric. You can get that field with getAddress and you can set that field (a dangerous operation) with setAddress. You can test to see if that field points to a null pointer with the isNull method. Finally, you can see whether two fields are equal with isEqual (remember that the Java objects may be different, but if their Electric addresses are the same, then these are the same Electric object).

Database Examination

To begin a search through the database, it is important to know the current library. This is done with:

Electric.curLib()

which returns a pointer to a Library object (for example Library 15464800). From here, the current cell can be obtained with:

Electric.getVal(Electric.curLib(), "firstnodeproto")

Essentially, any attribute can be examined with getVal, and new attributes can be created with setVal.

getVal has the following format:

Electric.getVal(object, attribute)

where object is the object being accessed and attribute is the attribute being requested. A list of all existing attributes on the Electric objects is given at the end of this document.

New attributes can be created on any object with setVal. In general, many of the existing attributes that are described at the end of this document cannot be set with setVal, but rather are controlled with special database modification methods. The format for setVal is:

Electric.setVal(object, attribute, value, options)

Where the options are either 0 or vdisplay to show this attribute when displaying the object. For example, to add a new attribute called "power-consumption" to the transistor component "t1", and give it the value 75, use:

Electric.setVal(t1, "power-consumption", 75, 0);

To add a displayed name to node "t1", use:

Electric.setVal(t1, "NODE name", "O1", Electric.vdisplay);

You can set arrays of values as well. For example, to set the shape of pure-layer node "metal" to be a diamond, use:

Integer[] outline = new Integer[8];



```

outline[0] = -1000; outline[1] = 0;
outline[2] = 0; outline[3] = 1000;
outline[4] = 1000; outline[5] = 0;
outline[6] = 0; outline[7] = -1000;
Electric.setVal(metal, "trace", outline, 0);

```

Single entries in array attributes can be set, with:

```
Electric.setInd(object, attribute, index, value)
```

where index is the 0-based entry in the array.

Finally, attributes can be deleted with:

```
Electric.delVal(object, attribute)
```

However, only those attributes that have been created with setVal can be deleted in this way. The other attributes are protected.

Basic Synthesis

To create a new cell in the current library, use:

```
Electric.newNodeProto(cellName, Electric.curLib())
```

which returns a NodeProto pointer that can be used in subsequent calls which place components and wires in that cell.

To get the address of an existing NodeProto, use:

```
Electric.getNodeProto(cellName)
```

which returns the same type of value as newNodeProto. Thus, the code:

```
Electric.NodeProto mycell = Electric.newNodeProto("adder{lay}", Electric.curLib());
```

is the same as the code:

```
Electric.newNodeProto("adder{lay}", Electric.curLib());
```

```
Electric.NodeProto mycell = Electric.getNodeProto("adder{lay}");
```

and both deal with the "layout" view of the cell called "adder".

To create a component in a cell, use:

```
Electric.newNodeInst(proto, lowX, highX, lowY, highY, transpose, angle, cell)
```

where proto is a NodeProto of the component that is to be created, lowX, highX, lowY, and highY are the bounds of the component, angle is the number of tenth-degrees of rotation for the component, transpose is nonzero to transpose the component's orientation (after rotation), and cell is the NodeProto in which to place the component.

The four bounds values are somewhat confusing to compute. For primitive components (such as Transistors), any value is acceptable and the component will scale. However, it is still nice to know the default value, which can be obtained from the NodeProto with getVal as follows:

```
Electric.NodeProto tran = Electric.getNodeProto("P-Transistor");
```

```
int lowx = ((Integer)Electric.getVal(tran, "lowx")).intValue();
```

```
int highx = ((Integer)Electric.getVal(tran, "highx")).intValue();
```

```
int lowy = ((Integer)Electric.getVal(tran, "lowy")).intValue();
```

```
int highy = ((Integer)Electric.getVal(tran, "highy")).intValue();
```

When complex components (cells) are placed, the bounds **MUST** be exactly the same as the bounding box of the cell's contents. This information is available in the above manner. As an example of newNodeInst, and given the above bounds calculations, a default size P-Transistor is created in cell "adder" with:

```
Electric.NodeInst t1 = Electric.newNodeInst(tran, lowx, highx, lowy, highy, 0, 0, mycell);
```

The returned pointer to the transistor component will be used later when wiring.



To wire two components, it is necessary to know these four things:

- The component objects on the two ends, returned by `newNodeInst`
- The `PortProto` values of the connection sites on the components
- The X and Y coordinates of the connection sites
- The type, width, and other characteristics of the wire being created

Connection sites are called `PortProtos` and are associated with `NodeProtos`. To get the address, use:

```
Electric.getPortProto(nodeProto, portName)
```

For example, to get the polysilicon port on the left side of the MOSIS CMOS P-Transistor, use:

```
Electric.PortProto polyleft = Electric.getPortProto(tran, "p-trans-poly-left");
```

Unfortunately, there is no good way to get a list of port names on the primitive components. There are, however, some simplifications. For example, if there is only one port (as is the case with most contacts and pins) then its name is not necessary:

```
Electric.PortProto port = (Electric.PortProto)Electric.getVal(tran, "firstportproto");
```

This will obtain the first port on the P-Transistor component. To obtain the coordinates of a port for wiring, use

```
Electric.portPosition(node, port)
```

This returns an array with the coordinates. For example:

```
Integer[] portpos = Electric.portPosition(t1, polyleft);
```

will obtain the coordinate of the "p-trans-poly-left" port on the newly created P-Transistor, t1. The X value will be `portpos[0].intValue()` and the Y value will be `portpos[1].intValue()`.

The final piece of information necessary is the type of arc and the width of the arc. Given an arc name, the type can be obtained with:

```
Electric.getArcProto(arcName)
```

Given an `ArcProto`, its default width can be obtained with:

```
Electric.getVal(arc, "nominalwidth")
```

When all of the information is ready, the call:

```
Electric.newArcInst(arcType, width, bits, nodeA, portA, xA, yA, nodeB, portB, xB, yB, cell)
```

places the wire. You can ignore the value of `bits` and set it to zero.

The size used to create a node or arc is not necessarily the size of the object. This is because the size given to `newNodeInst` is the outer bounds of the object which may include implant layers. To get the difference between the "nominal size" and the `newNodeInst` size, use:

```
Electric.nodeProtoSizeOffset(primitiveNodeProto)
```

which returns an array of 4 Integers with the low X, high X, low Y and high Y offsets. The routine:

```
Electric.arcProtoWidthOffset(arcProto)
```

returns the difference between the "nominal width" and the actual width used in `newArcInst`.

Here is a complete example of placing a transistor, a contact, and running a wire between them (the result is shown at the bottom).

```
/* create a cell called "tran-contact" in the current library */
```

```
Electric.NodeProto mycell = Electric.newNodeProto("tran-contact", Electric.curLib());
```

```
/* get pointers to primitives */
```

```
Electric.NodeProto tran = Electric.getNodeProto("P-Transistor");
```

```
Electric.NodeProto contact = Electric.getNodeProto("Metal-1-Polysilicon-1-Con");
```

```
/* get default sizes of these primitives */
```

```
int tlowx = ((Integer)Electric.getVal(tran, "lowx")).intValue();
```

```
int thighx = ((Integer)Electric.getVal(tran, "highx")).intValue();
```



```

int tlowy = ((Integer)Electric.getVal(tran, "lowy")).intValue();
int thighy = ((Integer)Electric.getVal(tran, "highy")).intValue();
int clowx = ((Integer)Electric.getVal(contact, "lowx")).intValue();
int chighx = ((Integer)Electric.getVal(contact, "highx")).intValue();
int clowy = ((Integer)Electric.getVal(contact, "lowy")).intValue();
int chighy = ((Integer)Electric.getVal(contact, "highy")).intValue();

/* get pointer to Polysilicon arc and its default width */
Electric.ArcProto arctype = Electric.getArcProto("Polysilicon-1");
int width = ((Integer)Electric.getVal(arctype, "nominalwidth")).intValue();

/* create the transistor and the contact to its left */
Electric.NodeInst c1 = Electric newNodeInst(contact, clowx, chighx, clowy, chighy,
0, 0, mycell);
Electric.NodeInst t1 = Electric newNodeInst(tran, tlowx+8000, thighx+8000,
tlowy, thighy, 0, 0, mycell);

/* get the transistor's left port coordinates */
Electric.PortProto tport = Electric.getPortProto(tran, "p-trans-poly-left");
Integer[] tpos = Electric.portPosition(t1, tport);

/* get the contacts's only port coordinates */
Electric.PortProto cport = (Electric.PortProto)Electric.getVal(contact, "firstportproto");
Integer[] cpos = Electric.portPosition(c1, cport);

/* run a wire between the primitives */
Electric.newArcInst(arctype, width, 0,
t1, tport, tpos[0].intValue(), tpos[1].intValue(),
c1, cport, cpos[0].intValue(), cpos[1].intValue(), mycell);

```



Hierarchy

Cells, as created by `newNodeProto`, can be placed in other cells with `newNodeInst`. The instances simply use complex `NodeProto` fields rather than primitive `NodeProtos` as in the above example. For example, the following code creates a new cell called "two-trans" and places two instances of the above "tran-contact" cell, one above the other.



```

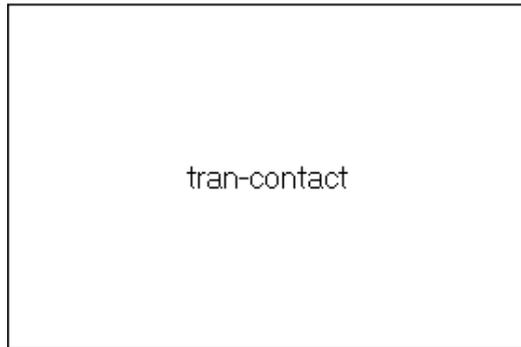
/* create a cell called "two-trans" */
Electric.NodeProto highercell =
Electric newNodeProto("two-trans",
Electric.curLib());

/* get pointer to the "tran-contact" cell */
Electric.NodeProto tc =
Electric.getNodeProto("tran-contact");

/* get size of this cell */
int lowx = ((Integer)Electric.getVal(tc,
"lowx")).intValue();
int highx = ((Integer)Electric.getVal(tc,
"highx")).intValue();
int lowy = ((Integer)Electric.getVal(tc,
"lowy")).intValue();
int highy = ((Integer)Electric.getVal(tc,
"highy")).intValue();

/* create two cell instances, one above the
other */
Electric.NodeInst o1 =
Electric newNodeInst(tc,
lowx, highx, lowy, highy, 0, 0, highercell);
Electric.NodeInst o2 =
Electric newNodeInst(tc,
lowx, highx, lowy+10000, highy+10000,
0, 0, highercell);

```



Another necessary feature, when making hierarchy, is the ability to place wires between connection sites on cell instances. To do this, it is necessary to create exports. This takes a port on a primitive component (for example, the transistor or contact in the "tran-contact" cell) and makes it into an export on the current cell. This is done with:

```
Electric.newPortProto(cell, nodeInCell, portOnNode, portName)
```

where cell is the cell containing the component whose port is being exported, nodeInCell is that component, and portOnNode is the particular port on that node being exported. For example, to export the top and bottom diffusion ports in the "tran-contact" cell (as shown here), the following code can be added:

```

Electric.newPortProto(myCell, t1,
Electric.getPortProto(tran,
"p-trans-diff-top"),
"topdiff");
Electric.newPortProto(myCell, t1,
Electric.getPortProto(tran,
"p-trans-diff-bottom"),
"botdiff");

```



And then, the components "o1" and "o2" in the cell "two-trans" can be wired, using the ports called "topdiff" and "botdiff":



```

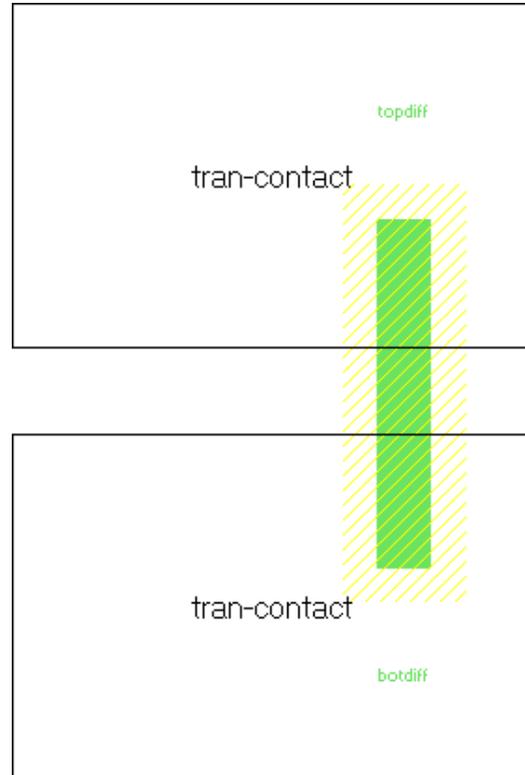
/* get pointer to P-Active arc and its default
width */
Electric.ArcProto darctype =
  Electric.getArcProto("P-Active");
int dwidth =
  ((Integer)Electric.getVal(darctype,
  "nominalwidth")).intValue();

/* get the bottom cell's top port */
Electric.PortProto lowport =
  Electric.getPortProto(tc,
  "topdiff");
Integer[] lowpos = Electric.portPosition(o1,
  lowport);

/* get the top cell's bottom port */
Electric.PortProto highport =
  Electric.getPortProto(tc,
  "botdiff");
Integer[] highpos = Electric.portPosition(o2,
  highport);

/* run a wire between the primitives */
Electric.newArcInst(darctype, dwidth, 0,
  o1, lowport, lowpos[0].intValue(),
  lowpos[1].intValue(),
  o2, highport, highpos[0].intValue(),
  highpos[1].intValue(), highercell);

```



Another set of routines exists for hierarchy traversal. When generating a netlist from a circuit, the hierarchy is typically traversed down to the bottom. Cell parameters are evaluated based on the traversal path to the particular cell with the parameter. To inform the system of the traversal path, it is necessary to call special routines during traversal.

At the start of traversal, call

```
Electric.beginTraverseHierarchy()
```

Before descending into a cell instance, call

```
Electric.downHierarchy(node, index)
```

where node is the cell instance node, and index is the index of which node (if it is arrayed).

After returning from the examination of the cell instance, call

```
Electric.upHierarchy()
```

Finally, after completing hierarchy traversal, call

```
Electric.endTraverseHierarchy()
```

To find out your location in the hierarchy (if this object is being viewed inside another) use:

```
Electric.getTraversalPath()
```

which returns an array of NodeInsts, terminated by a null one. Each entry is the instance, higher up the hierarchy, that contains the former level.



Modification

Two types of modification can be done to existing objects: deletion and change. To delete a cell, use:

Electric.killNodeProto(cell)

To make a copy of a cell (within the same library or from one library to another), use:

Electric.copyNodeProto(fromCell, toLibrary, toCellName)

where fromCell is the original cell (NodeProto) and toLibrary is the destination library. Use curLib to copy to the same library. The new cell name is the last parameter. The method returns the address of the new cell (NodeProto).

To delete a component, use:

Electric.killNodeInst(node)

Before a component can be deleted, all wires and exports must be removed.

To change the size or orientation of a component, use:

Electric.modifyNodeInst(node, dLowX, dLowY, dHighX, dHighY, dRotation, dTrans)

where dLowX, dLowY, dHighX, and dHighY are the changes to position and size. dRotation and dTrans are changes to the orientation.

To change the prototype of a component, use:

Electric.replaceNodeInst(oldNode, newPrototype)

where the old component is oldNode, and the new NodeProto that should be in its place is newPrototype. This new prototype must be able to connect to all existing arcs. The method returns the address of the new component.

To delete a wire, use:

Electric.killArcInst(arc)

To change the width or position of a wire, use:

Electric.modifyArcInst(arc, dWidth, dX1, dY1, dX2, dY2)

where dWidth, dX1, dY1, dX2, and dY2 are the changes to the width, X/Y position of end 1, and X/Y position of end 2. Note that position changes cannot cause the connecting nodes to move, so the changes may only be small ones that work within the ports.

To change the prototype of a wire, use:

Electric.replaceArcInst(oldArc, newPrototype)

where oldArc is the former wire and newPrototype is the new ArcProto to use. The nodes on either end must be able to accept this new type of wire. The method returns the address of the new wire.

To delete an export, use:

Electric.killPortProto(cell, port)

which will remove port port on cell cell.

To move an export from one component to another (keeping connected wires), use:

Electric.movePortProto(cell, oldPort, newNode, portOnNewNode)

where the old port is oldPort in cell cell, and it is now moved to component newNode (which is also in cell cell), port portOnNewNode of that component.



Search

A common operation is a search of all components in a cell. The following code prints the name of all components in the cell "mycell":

```
Electric.NodeInst node;  
for(node = (Electric.NodeInst)Electric.getVal(mycell, "firstnodeinst");  
!node.isNull();  
node = (Electric.NodeInst)Electric.getVal(node, "nextnodeinst"))  
{  
System.out.println("Found " + describeNode(node) + " node");  
}
```

Where describeNode is defined as follows (the name of a node is found in different places depending on whether it is a primitive or complex NodeProto):

```
public static String describeNode(Electric.NodeInst node)  
{  
Electric.NodeProto proto = (Electric.NodeProto)Electric.getVal(node, "proto");  
if (((Integer)Electric.getVal(proto, "primindex")).intValue() != 0)  
return((String)Electric.getVal(proto, "primname"));  
return((String)Electric.getVal((Electric.Cell)Electric.getVal(proto, "cell"), "cellname"));  
}
```

And the following code prints the name of all wires in the cell "mycell":

```
Electric.ArcInst arc;  
for(arc = (Electric.ArcInst)Electric.getVal(mycell, "firstarcinst");  
!arc.isNull();  
arc = (Electric.ArcInst)Electric.getVal(arc, "nextarcinst"))  
{  
String arcname = (String)Electric.getVal((Electric.ArcProto)Electric.getVal(arc, "proto"),  
"protoname");  
System.out.println("Found " + arcname + " arc");  
}
```

To do a search of all nodes and arcs in a rectangular area of a cell, first call:

```
Electric.initSearch(lowX, highX, lowY, highY, cell)
```

where lowX, highX, lowY, and highY are the coordinates to search in cell cell (a NodeProto). This method will return an integer search key that can then be passed repeatedly to:

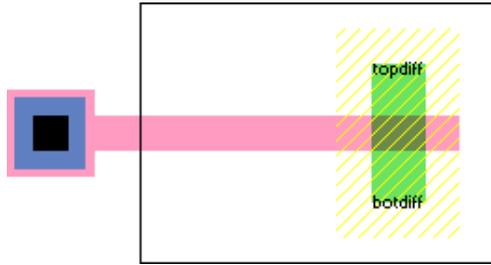
```
Electric.nextObject(searchKey)
```

which will return Geom objects of each node and arc in the search area. When this method returns a null Geom, the search is complete. Geom objects can point to either nodes or arcs, depending on their "entryisnode" attribute. Then, the "entryaddr" attribute will point to the actual NodeInst or ArcInst. If you wish to terminate the search early, call:

```
Electric.termSearch(searchKey)
```

Here is an example of code that prints the names of all nodes and arcs in the area ($2000 \leq X \leq 10000$, $-3000 \leq Y \leq 3000$). The selected area is shown as a black box here.





```

int key = Electric.initSearch(2000, 10000, -3000, 3000, mycell);
for(;;)
{
    Electric.Geom object = Electric.nextObject(key);
    if (object.isNull()) break;
    int isnode= ((Integer)Electric.getVal(object, "entryisnode")).intValue();
    if (isnode != 0)
    {
        Electric.NodeInst ni = (Electric.NodeInst)Electric.getVal(object, "entryaddr");
        System.out.println("Found node " + describenode(ni));
    } else
    {
        Electric.ArcInst ai = (Electric.ArcInst)Electric.getVal(object, "entryaddr");
        String arcname = (String)Electric.getVal((Electric.ArcProto)Electric.getVal(ai, "proto"),
        "protoname");
        System.out.println("Found arc " + arcname);
    }
}

```

Layers and Polygons

Nodes and arcs are built out of layers, and layers are described with objects of type Polygon: To get all of the layers in a node, first call:

Electric.nodePolys(node)

to get the number of polygons on the node, and then make repeated calls to:

Electric.shapeNodePoly(node, index)

to get the polygons.

If you wish to get the electrical layers (a larger set that breaks layers where they cross an electrical boundary), use nodeEPolys and shapeENodePoly. To get all of the layers in an arc, first call arcPolys to get the number of polygons on the arc, and then make repeated calls to shapeArcPoly to get the polygons.

Because polygon objects are created dynamically, they must be freed when you are done with them. Call:

Electric.freePolygon(poly)

to deallocate a Polygon.

To get information about a particular layer in a technology, call

Electric.layerName(tech, layer)

to get its name or

Electric.layerFunction(tech, layer)

to get its behavior (as described in the module "efunction.h").

The following example finds all polygons on a node and prints their layer names:



```

for(node = (Electric.NodeInst)Electric.getVal(myCell, "firstnodeinst");
   !node.isNull();
   node = (Electric.NodeInst)Electric.getVal(node, "nextnodeinst"))
{
    int polys = Electric.nodePolys(node);
    for(int i = 0; i < polys; i++)
    {
        Electric.Polygon poly = Electric.shapeNodePoly(node, i);
        Electric.Technology tech = (Electric.Technology)Electric.getVal(poly, "tech");
        int layer = ((Integer)Electric.getVal(poly, "layer")).intValue();
        int count = ((Integer)Electric.getVal(poly, "count")).intValue();
        String layerName = Electric.layerName(tech, layer);
        System.out.println("Polygon on layer " + layerName + " has " + count + " points");
        Electric.freePolygon(poly);
    }
}

```

There are three routines available to get design rules for layers.

Electric.maxDRCSurround(tech, library, layer)

returns the maximum distance around the layer (in the specified technology and library) that any design rule can be. The routine:

Electric.DRCMinDistance(tech, library, layer1, layer2)

returns the minimum distance between the layers (in the specified technology and library). The routine:

Electric.DRCMinWidth(tech, library, layer)

returns the minimum feature size of the layer (in the specified technology and library).

Because nodes may be rotated, it is necessary to apply the node's transformation to all polygons before using the coordinate values. To create the transformation that accounts for a node's rotation, use:

Electric.makeRot(node)

which returns an XArray object. To create the transformation that accounts for the node's position within its parent, use:

Electric.makeTrans(node)

To apply a transformation to a polygon, use:

Electric.xformPoly(poly, trans)

Views

A view is an object that describes a cell. There are many standard views: Layout, Schematic, Icon, Simulation-snapshot, Skeleton, VHDL, Verilog, Document, Unknown, and many flavors of Netlist. In addition, new views can be created with "newView":

Electric.newView(viewName, abbreviation)

and views can be deleted with killView (the standard views cannot be deleted):

Electric.killView(view)

To get a view object, use getView on its name.

To associate different views of a cell, the methods iconView and contentsView obtain different cells. For example:

Electric.iconView(mycell)

finds the associated icon cell of the cell in which "mycell" resides.



Libraries

In the above examples, the current library was always used. This is determined by calling:

Electric.curLib()

However, there can be other libraries. To get a specific named library, use:

Electric.getLibrary(libName)

To create a new library, use:

Electric.newLibrary(libraryName, libraryFile)

where libraryName is the name to use, and libraryFile is the path name where this library will be saved. This method returns the address of a new library object that can then be used when creating cells.

Only one library is the current one, and to switch, you must use:

Electric.selectLibrary(lib)

A library can be deleted with:

Electric.killLibrary(lib)

A library can be erased (its cells deleted, but not the library) with:

Electric.eraseLibrary(lib)

Technologies

A technology is an environment of design that includes primitive components and wire prototypes. The current technology can be obtained with:

Electric.curTech()

A specific technology can be obtained from its name with:

Electric.getTechnology(techName)

All technologies can be found by traversing a linked list, the head of which is a technology named "Generic".

Tools

A tool is a piece of synthesis or analysis code that can operate upon the database. A particular tool object can be obtained with:

Electric.getAid(toolName)

where the possible names of tools are:

"compaction"	circuit compaction
"compensation"	geometry compensation
"drc"	design-rule checking
"erc"	electrical-rule checking
"io"	input/output control
"logeffort"	logical effort analysis
"network"	network maintenance
"pla"	programmable logic array generator



"project"	project management
"routing"	automatic wire routing
"silicon-compiler"	netlist-to-layout silicon assembler
"simulation"	simulation
"user"	the user interface
"vhdl-compiler"	VHDL-to-netlist compiler

The number of tools is available with:

Electric.maxAid()

And a particular tool, indexed from 0 to Electric.maxAid()-1 can be obtained with:

Electric.indexAid(index)

A tool can be switched on with:

Electric.toolTurnOn(tool)

where tool is a tool object.

A tool can be turned off with:

Electric.toolTurnOff(tool)

A tool can be given a specific instruction with:

Electric.tellTool(tool, count, parameters)

where the parameters is an array of count strings. For example, to list all technologies, use this code:

Electric.Tool user = Electric.getTool("user");

String[] message = new String[2];

message[0] = "show";

message[1] = "technologies";

Electric.tellTool(user, 2, message);

The command "show technologies" and other commands are from the low-level command interpreter, which is documented fully in the Internals Manual.

Miscellaneous

To force display changes to be drawn on the screen, you must use:

Electric.flushChanges()

Every change to the database is queued internally in a "batch" which includes the change and any constrained side-effects of that change. A new batch is created for each Java session with the interpreter (also for each Electric command that is issued from the keyboard/mouse). To reverse the last batch of changes, use:

Electric.undoABatch()

Multiple calls to this method in a single batch will undo multiple batches. To erase the list of change batches, use:

Electric.noUndoAllowed()

If you are creating a wire that makes many bends, it is necessary to create special nodes called "pins" at each bend. To find out what kind of pin to use for a given wire type, use:

Electric.getPinProto(arc)

where arc is the wire type, and the method returns the component type (NodeProto) of the pin.



Network objects can be obtained by name with the method `getNetwork` which takes a name and a cell in which to search. For example, the code:

```
Electric.getNetwork("insig", mycell)
```

obtains the address of the network called "insig" in cell `mycell`.

The generic function of a node instance can be determined with:

```
Electric.nodeFunction(node)
```

which returns a value from the list of constants in the C header file "efunction.h". This value is essentially the same one as would be obtained by looking at the "userbits" field of the node's prototype. However, certain components that have generic prototypes will be made more specific by this method.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 11: INTERPRETERS



11-5: Interpreter Attributes



When examining the database from the language interpreters, you have access to many attributes on the various objects. This section lists the predefined attributes on the different Electric objects. Those attributes with a (*) next to them are relatively important to database examination.

This section is not meant as a full explanation of the attributes in the Electric database, but rather is a quick list. For more detail on these and other aspects of Electric internals, see the Electric Internals Manual (a document that is available from [Static Free Software](#)).

These basic attributes exist on components (NODEINST):			
	ATTRIBUTE	TYPE	DESCRIPTION
	aseen	Integer	flags for the database
	firstportarcinst	PORTARCINST	head of linked list of connecting arcs' ports
	firstportexpinst	PORTEXPINST	head of linked list of exports
	geom	GEOM	geometry module
*	highx	Integer	high X coordinate in database units
*	highy	Integer	high Y coordinate in database units
	lastinst	NODEINST	link to previous component of this type
	lastnodeinst	NODEINST	link to previous component in this cell
*	lowx	Integer	low X coordinate in database units
*	lowy	Integer	low Y coordinate in database units
	nextinst	NODEINST	link to next component of this type
	nextnodeinst	NODEINST	link to next component in this cell
*	parent	NODEPROTO	cell that contains this component
*	proto	NODEPROTO	type of this component
*	rotation	Integer	angle in degrees of this component
*	transpose	Integer	nonzero if component transposed after rot.
*	userbits	Integer	miscellaneous flags



These basic attributes exist on component prototypes (NODEPROTO):			
	ATTRIBUTE	TYPE	DESCRIPTION
	adirty	Integer	flags for the database
*	cell	CELL	cell of which this cell is a part
*	cellview	VIEW	view of this cell
	creationdate	Integer	date this cell was created
	firstarcinst	ARCINST	head of list of wires in this cell
	firstinst	NODEINST	head of list of instances of this prototype
	firstnetwork	NETWORK	head of list of networks in this cell
	firstnodeinst	NODEINST	head of list of components in this cell
*	firstportproto	PORTPROTO	head of list of exports on this cell
*	highx	Integer	high X coordinate in database units
*	highy	Integer	high Y coordinate in database units
*	primindex	Integer	nonzero if this is a primitive prototype
	lastnodeproto	NODEPROTO	link to previous prototype in lib/tech
	lastversion	NODEPROTO	earlier version of this cell
*	lowx	Integer	low X coordinate in database units
*	lowy	Integer	low Y coordinate in database units
	newestversion	NODEPROTO	most recent version of this cell
	nextnodeproto	NODEPROTO	link to next prototype in lib/tech
	nextincell	NODEPROTO	next view in this cell
	revisiondate	Integer	date this cell was last modified
	rtree	RTNODE	root R-tree in this cell
*	primname	String	name of this primitives (if primitive)
*	tech	TECHNOLOGY	technology in which this primitive resides
*	userbits	Integer	miscellaneous flags
*	version	Integer	version number of this cell

These basic attributes exist on cells (CELL):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	cellname	String	name of this cell
*	firstincell	NODEPROTO	first cell in this cell
*	lib	LIBRARY	library containing this cell
*	nextcell	CELL	link to next cell in this library



These basic attributes exist on instantiated wire connections (PORTARCINST):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	conarcinst	ARCINST	wire that is connected at this port
	nextportarcinst	PORTARCINST	link to next instantiated wire connection
*	proto	PORTPROTO	prototype of the port that is connected

These basic attributes exist on instantiated export instances (PORTEXPINST):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	exportproto	PORTPROTO	export prototype on parent cell
	nextportexpinst	PORTEXPINST	link to next instantiated export connection
*	proto	PORTPROTO	prototype of the port that is an export

These basic attributes exist on connection prototypes (PORTPROTO):			
	ATTRIBUTE	TYPE	DESCRIPTION
	aseen	Integer	flags for the database
*	connects	ARCPROTO array	array of arc types that may connect
*	network	NETWORK	network object
	nextportproto	PORTPROTO	link to next connection prototype
*	parent	NODEPROTO	component prototype with connection
*	protoname	String	name of connection
*	subnodeinst	NODEINST	origin component in cell
*	subportexpinst	PORTEXPINST	origin export component in cell
*	subportproto	PORTPROTO	origin port on component in cell
*	userbits	Integer	miscellaneous flags

These basic attributes exist on wires (ARCINST):			
	ATTRIBUTE	TYPE	DESCRIPTION
	aseen	Integer	flags for the database
*	endshrink	Integer	data for nonmanhattan end shrinkage
*	geom	GEOM	geometry module
	lastarcinst	ARCINST	link to previous wire in cell
*	length	Integer	length in database units
*	network	NETWORK	network object
	nextarcinst	ARCINST	link to next wire in cell



*	nodeinst1	NODEINST	component on end 1
*	nodeinst2	NODEINST	component on end 2
*	parent	NODEPROTO	cell that contains this wire
*	portarcinst1	PORTARCINST	instantiated wire connection on end 1
*	portarcinst2	PORTARCINST	instantiated wire connection on end 2
*	proto	ARCPROTO	type of this wire
*	userbits	Integer	miscellaneous flags
*	width	Integer	width in database units
*	xpos1	Integer	X coordinate of end 1 in database units
*	xpos2	Integer	X coordinate of end 2 in database units
*	ypos1	Integer	Y coordinate of end 1 in database units
*	ypos2	Integer	Y coordinate of end 2 in database units

These basic attributes exist on wire prototypes (**ARCPROTO**):

	ATTRIBUTE	TYPE	DESCRIPTION
	arcindex	Integer	0-based index of this arc type
	nextarcproto	ARCPROTO	link to next arc type in this technology
*	nominalwidth	Integer	default wire width in database units
*	protoname	String	name of this wire type
*	tech	TECHNOLOGY	technology in which this wire type resides
*	userbits	Integer	miscellaneous flags

These basic attributes exist on networks (**NETWORK**):

	ATTRIBUTE	TYPE	DESCRIPTION
*	netname	String	name of this network
	namecount	Integer	number of names
	arccount	Integer	number of arcs on this network
	arcaddr	ARCINST array	address of arc(s) on this network
	refcount	Integer	number of arcs on network
	portcount	Integer	number of ports on this network
	buslinkcount	Integer	number of busses referencing this network
*	parent	NODEPROTO	cell that has this network
*	signals	Integer	width of bus or index into bus
*	networklist	NETWORK array	list of single-wire networks on bus
	nextnetwork	NETWORK	next in linked list



	lastnetwork	NETWORK	previous in linked list
--	-------------	----------------	-------------------------

These basic attributes exist on geometric objects (GEOM):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	entryisnode	Integer	nonzero for component
*	entryaddr	NODEINST or ARCINST	address of component or wire
*	highx	Integer	high X coordinate in database units
*	highy	Integer	high Y coordinate in database units
*	lowx	Integer	low X coordinate in database units
*	lowy	Integer	low Y coordinate in database units

These basic attributes exist on R–tree nodes (RTNODE):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	flag	Integer	nonzero if pointers are terminal (geom)
*	highx	Integer	high X coordinate in database units
*	highy	Integer	high Y coordinate in database units
*	lowx	Integer	low X coordinate in database units
*	lowy	Integer	low Y coordinate in database units
*	parent	RTNODE	parent R–tree node
*	pointers	RTNODE array	children (type depends on "flag")
*	total	Integer	number of children in this node

These basic attributes exist on cell libraries (LIBRARY):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	curnodeproto	NODEPROTO	currently edited cell in library
	firstcell	CELL	head of list of cells in library
	firstnodeproto	NODEPROTO	head of list of cells in library
	lambda	Integer array	values of lambda for all technologies
*	libname	String	name of this library
*	libfile	String	disk file associated with this library
	nextlibrary	LIBRARY	link to next library in Electric
*	userbits	Integer	miscellaneous flags



These basic attributes exist on design environment objects (TECHNOLOGY):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	deflambda	Integer	value of lambda in database units
	firstarcproto	ARCPROTO	head of list of wire types in this technology
	firstnodeproto	NODEPROTO	head of list of primitive components
	techindex	Integer	0-based index of this technology
*	nexttechnology	TECHNOLOGY	link to next technology in Electric
*	techdescript	String	long description of this technology
*	techname	String	short name of this technology
	userbits	Integer	miscellaneous flags (none at present)

These basic attributes exist on views (VIEW):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	nextview	VIEW	link to next view
*	viewname	String	name of this view
*	sviewname	String	abbreviated name of this view

These basic attributes exist on editing window partition objects (WINDOWPART):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	curnodeproto	NODEPROTO	cell in window
*	gridx	Integer	X spacing of grid
*	gridy	Integer	Y spacing of grid
	lastwindowpart	WINDOWPART	last in linked list
*	location	String	name of window
	nextwindowpart	WINDOWPART	next in linked list
	screenlx	Integer	low X coordinate in cell
	screenly	Integer	low Y coordinate in cell
	screenhx	Integer	high X coordinate in cell
	screenhy	Integer	high Y coordinate in cell
	state	Integer	miscellaneous information about window
	uselx	Integer	low X coordinate on screen
	usely	Integer	low Y coordinate on screen
	usehx	Integer	high X coordinate on screen
	usehy	Integer	high Y coordinate on screen



These basic attributes exist on graphical attribute objects (GRAPHICS):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	bits	Integer	bitplanes of color display
*	col	Integer	color to use
*	raster	Integer array	16x16 bit pattern

These basic attributes exist on editing constraint objects (CONSTRAINT):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	conname	String	name of constraint system
*	condesc	String	description of constraint system

These basic attributes exist on synthesis and analysis tools (AID):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	aidname	String	name of this tool
*	aidstate	Integer	miscellaneous flags
*	aidindex	Integer	0-based index of this tool

These basic attributes exist on polygons (POLYGON):			
	ATTRIBUTE	TYPE	DESCRIPTION
*	count	Integer	number of points
*	desc	GRAPHICS	graphic appearance
*	font	Integer	font size
*	layer	Integer	layer number
	limit	Integer	max allocated points
*	portproto	PORTPROTO	port association
*	string	String	message (if text style)
*	style	Integer	style
*	tech	TECHNOLOGY	technology
*	xv	Integer array	X coordinate values
*	yv	Integer array	Y coordinate values

[← Previous](#)

[↑ Table of Contents](#)

[Next →](#)

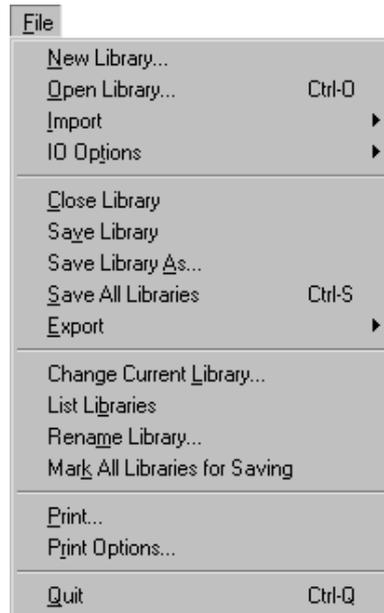




Chapter 12: MENU SUMMARY



12-1: The File Menu



This menu allows you to manipulate libraries of cells, which are typically read and written at one time. Besides basic library manipulation, this menu permits the reading and writing of libraries in various formats including CIF (Caltech Intermediate Format), GDS II (Calma's stream format), EDIF (the Electronic Design Interchange Format), and PostScript (a popular printer interface format).

New Library... [3-9]

This command creates a new library with no cells. It becomes the current library. Use the **Edit Cell...** command from the **Cells** menu to create circuitry.

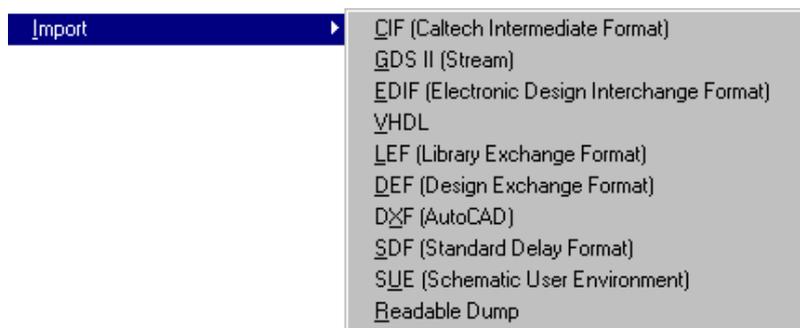
Open Library... [3-9]

This command reads a library of cells from disk. A dialog helps you select the file. Multiple libraries may be in memory at one time.



Import [3–9]

These commands allow a library to be read in a foreign interchange format. The possible file formats are shown in the submenu. For CIF, GDS II, and DXF, design–rule checking is disabled in order to avoid the inevitable errors that exist in the unconnected geometry.



CIF	This command reads a Caltech Intermediate Format (CIF) file from disk and creates a new library to contain the circuitry. Note that CIF contains no connectivity information, so the new library will have only polygonal information and no topology. To determine the CIF layer names that will correspond to those in the current technology, use the CIF Options... subcommand of the IO Options command.
GDS II	This command reads a Stream (GDS II) file from disk and creates a new library to contain the circuitry. On Windows, you can select multiple GDS files, and they will all be read into the current library. Note that GDS contains no connectivity information, so the new library will have only polygonal information and no topology. To determine the GDS layer numbers that will correspond to those in the current technology, use the GDS Options... subcommand of the IO Options command.
EDIF	This command reads an Electronic Design Interchange Format (EDIF) file from disk and creates a new library to contain the circuitry.
VHDL	This command reads a VHDL file from disk and creates a new library to contain the circuitry.
LEF	This command reads a Library Exchange Format (LEF) file from disk and creates a new library to contain the circuitry.
DEF	This command reads a Design Exchange Format (DEF) file from disk and creates a new library to contain the circuitry.



DXF

This command reads an AutoCAD DXF file from disk and creates a new library to contain the layout. Note that DXF contains no connectivity information, so the new library will have only polygonal information and no topology. You must switch to the "Artwork" technology before reading DXF, and should use the **DXF Options...** subcommand of the **IO Options** command to set layer associations.

SDF

This command reads a Standard Delay Format (SDF) file from disk and annotates the current library with the test vector information. Before this data can be used by the simulator, one of the three sets (**Typical**, **Minimum**, or **Maximum**) must be selected with the **Annotate Delay Data (ALS)** subcommand of the **Simulate** command of the **Tools** menu.

SUE

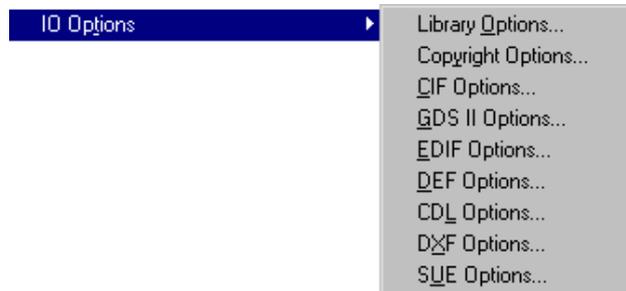
This command reads a Schematic User Environment (SUE) file from disk and adds the circuitry to the current library.

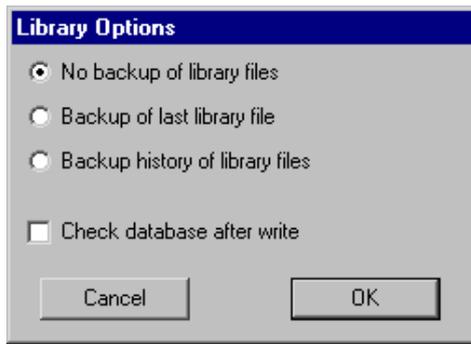
Readable Dump

Readable Dump files contain all of the information in an Electric library, except that they are ASCII (readable). This command reads such a file, allowing libraries to easily transport from other computers.

IO Options [3-9], [7-3]

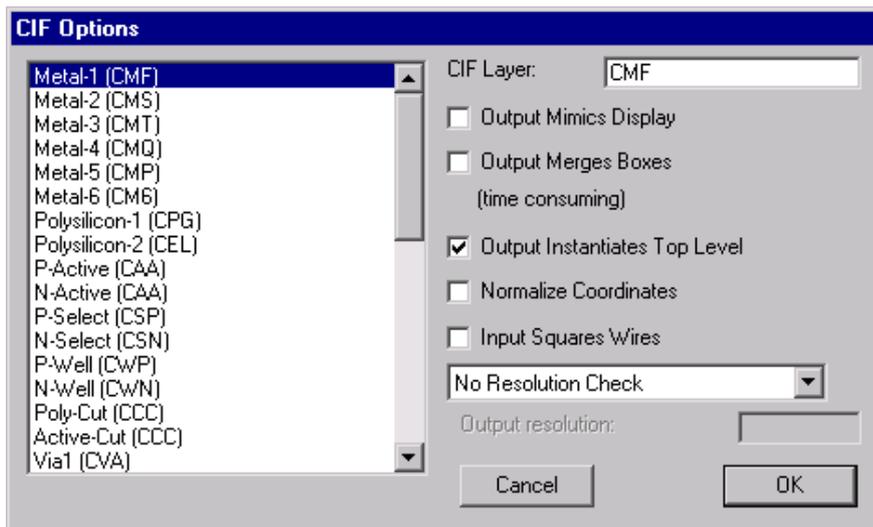
This command allows you to control the way that libraries are written, and also controls external formats such as CIF, GDS II, EDIF, DEF, CDL, DXF, and SUE.





The **Library Options...** command allows you to control whether backups are kept when libraries are saved. You can choose to backup the last library (which renames it so that it has a "~" at the end) or to backup a history of library files (which renames the former library file so that it has its creation date as part of its name). You can also request that the database be checked for consistency at write time.

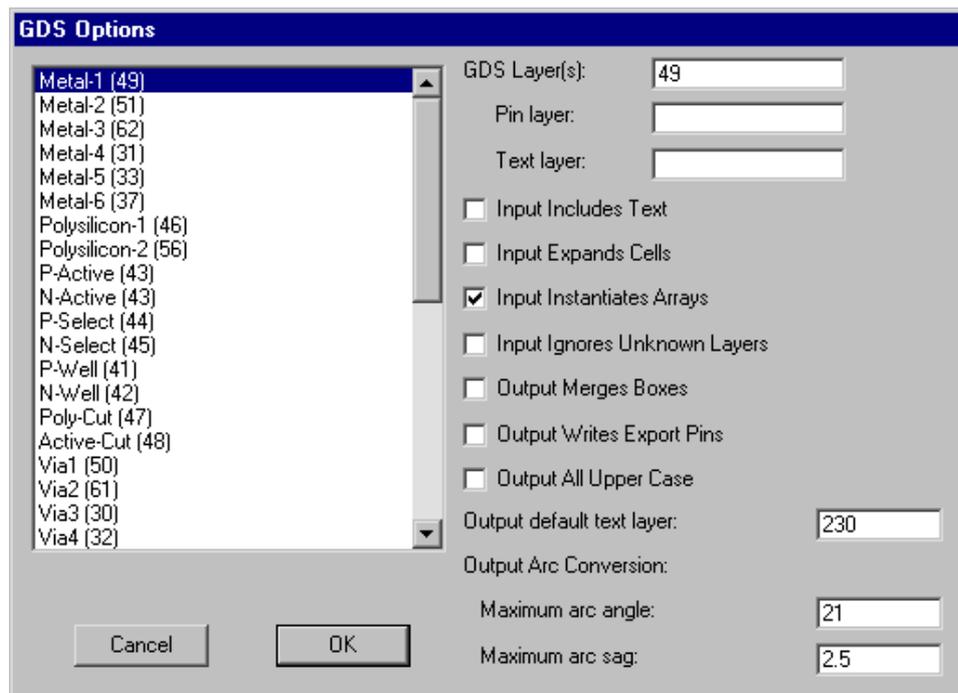
The **Copyright Options...** command allows you to place personalized copyright information into the files generated by Electric. The disk file with the copyright information should not contain any "comment" characters because they vary with each generated file format. Instead, Electric adds the proper comment characters for each output file format.



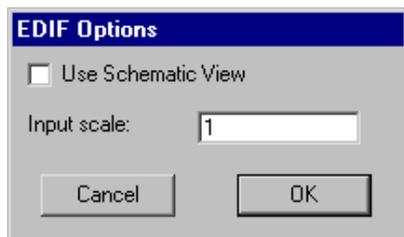
The **CIF Options...** command allows you to control CIF layer associations, and many other aspects of reading and writing CIF.



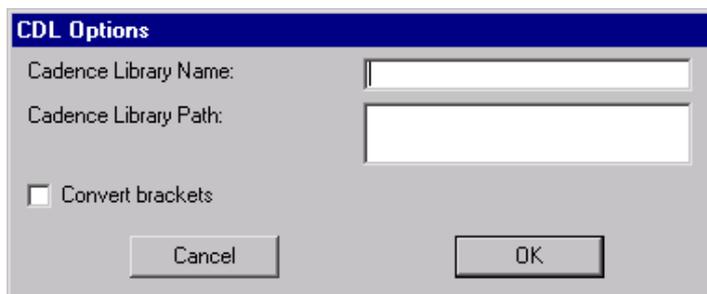
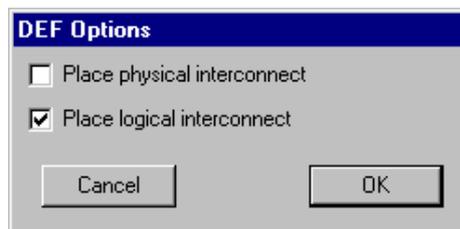
The **GDS Options...** command allows you to control GDS layer associations, and many other aspects of reading and writing GDS.



The **EDIF Options...** command allows you to control whether EDIF output writes the schematic or netlist view. It also controls the scale of EDIF input.



The **DEF Options...** command allows you to control which type of interconnect is written to the DEF file.

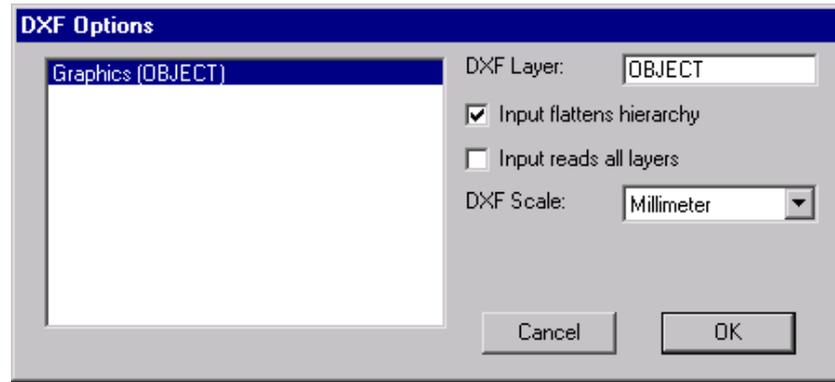


The **CDL Options...** command controls CDL output by providing library information and conversion options.



The DXF

Options... command allows you to control DXF layer associations, and whether DXF input has its hierarchy preserved or flattened, whether all layers are read, and the scale factor to use during input.



The **SUE Options...** command controls transistor conversion during SUE input.



Close Library [3-9]

This command deletes the current library. Because there must always be a library, this operation will not work if only one exists.

Save Library [3-9]

This command writes the current library to disk. Changes become permanent at this point.

Save Library As... [3-9]

This command allows the current library to be renamed before being written to disk. Both the library and the disk file are renamed.

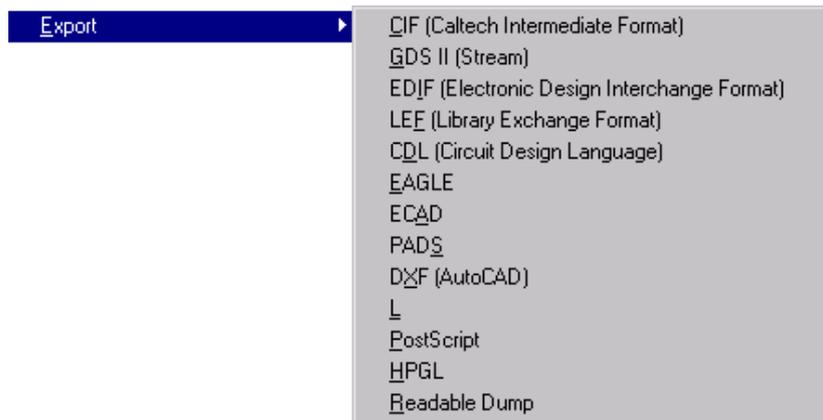
Save All Libraries [3-9]

This command writes all changed libraries to disk. Changes become permanent at this point.

Export [3-9]



These commands write all or part of the current library to disk in a foreign interchange format. The possible formats are shown in the submenu.



- | | |
|---------------|--|
| CIF | This command writes a CIF (Caltech Intermediate Format) description of the current cell to disk. The description includes everything below it in the hierarchy. To determine the CIF layer names that will be used, use the CIF Options... subcommand of the IO Options command. |
| GDS II | This command writes a Calma GDS II description of the current cell to disk. The description includes everything below it in the hierarchy. To determine the GDS layer numbers that will be used, use the GDS Options... subcommand of the IO Options command. |
| EDIF | This command writes an EDIF (Electronic Design Interchange Format) description of the current cell to disk. The description includes everything below it in the hierarchy. It contains connectivity only, and no layout. |
| LEF | This command writes a LEF (Library Exchange Format) description of the current cell to disk. The description includes everything below it in the hierarchy. It contains connectivity only, and no layout. |
| CDL | This command writes a CDL (a Cadence netlist format) description of the current cell to disk. The description includes everything below it in the hierarchy. It contains connectivity only, and no layout. |
| EAGLE | This command writes an EAGLE description of the current cell to disk. The description includes everything below it in the hierarchy. It contains connectivity only, and no layout. |
| ECAD | This command writes an ECAD description of the current cell to disk. The description includes everything below it in the hierarchy. It contains connectivity only, and no layout. |



PADS	This command writes a PADS description of the current cell to disk. The description includes everything below it in the hierarchy. It contains connectivity only, and no layout.
DXF	This command writes an AutoCAD DXF description of the current cell to disk. The description includes everything below it in the hierarchy. It contains geometry only, and no connectivity.
L	This command writes a description of the current cell in the "L" language, readable by GDT design tools.
PostScript	This command writes a description of the current cell in the "PostScript" language, a graphics format usable by many printers
HPGL	This command writes a description of the current cell in the "HPGL" language, a graphics format usable by many printers.
Readable Dump	This command writes the current library in an ASCII format that can be ported to other computers and still read into Electric.

Change Current Library... [3-9]

This command selects an existing library as the current one. It must have been read from disk.

List Libraries [3-9]

This command lists all libraries currently read in, and all dependencies between them.

Rename Library... [3-9]

This command allows you to rename a library.

Mark All Libraries for Saving [3-9]

This command requests that all libraries be saved to disk when the next **Save All Libraries** is done.

Print... [4-9]

This command causes the current cell to be printed. On UNIX systems, a temporary PostScript file is created and spooled to the printer. Another way to print is to use the **Write PostScript** and **Write HPGL** options of the **Export** command above.



Print Options... [4-9]

This command provides a number of options for printing, including a choice of what part of the circuit gets printed, whether the date is included, and which printer to use (UNIX only). For PostScript output, you can control encapsulation, color, paper size, and image rotation. PostScript output also allows you to set parameters for each cell: synchronization to a file and EPS scale. For HPGL output, you can control scale setting and HPGL version.

Printing Options

Plot Entire Cell Plot Date In Corner

Plot only Highlighted Area

Plot only Displayed Window

Default printer: <<Cannot set>>

Print and Copy resolution factor: 1

PostScript: Encapsulated Black&White

Printer Plotter

Width (in): 8.5 Height (in): 11 Margin (in): 0.74

No Rotation

For cell: noname:MyCircuit{lay}

EPS Scale: 2

Synchronize to file:

HPGL Level: HPGL HPGL/2

HPGL/2 plot fills page

HPGL/2 plot fixed at: 20 internal units per pixel

Quit

This command exits Electric. If libraries have been changed but not written to disk, you will be prompted to make sure that you want to exit the program.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 12: MENU SUMMARY



12-2: The Edit Menu



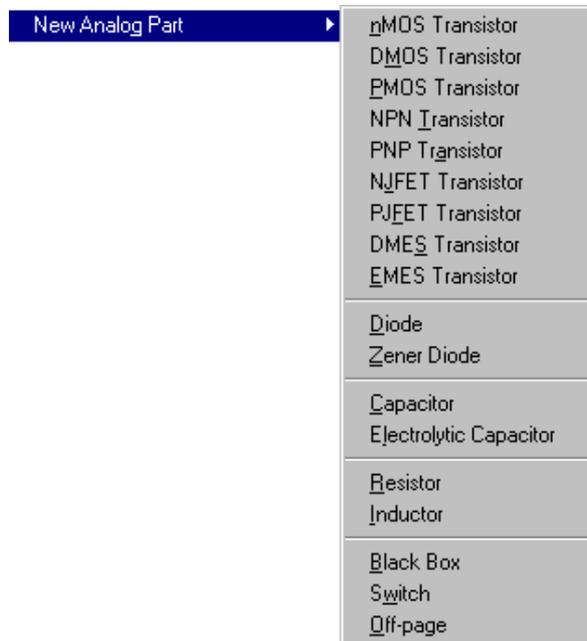
This menu provides all of the basic commands for creating, deleting, and modifying circuitry. Many special objects that do not appear in the component menu on the left can be created with these commands.

New Cell Instance... [3-3]

This command allows the creation of an instance of a cell. A dialog box will be presented to choose the desired cell, after which you can click in the editing window to create the instance.

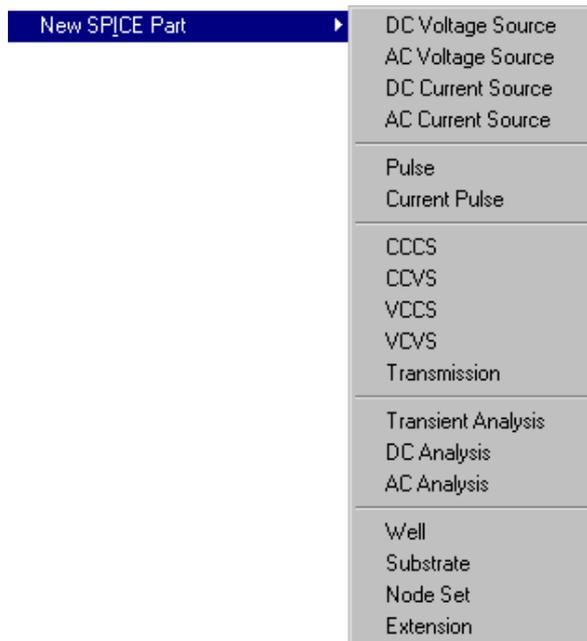


New Analog Part [7-6]



This command creates a schematic component from the submenu list. Many of these components use values which you will be asked to provide. For example, if you choose the Resistor, a dialog box will request a resistance value to display on that component.

New SPICE Part [9-4] [7-6]

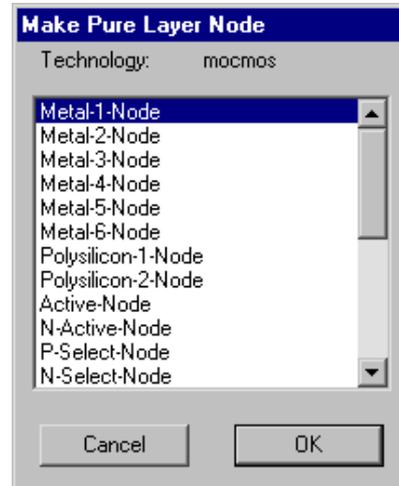


This command creates a SPICE simulation component from the submenu list. Many of these components use SPICE deck fragments which you will be asked to provide.

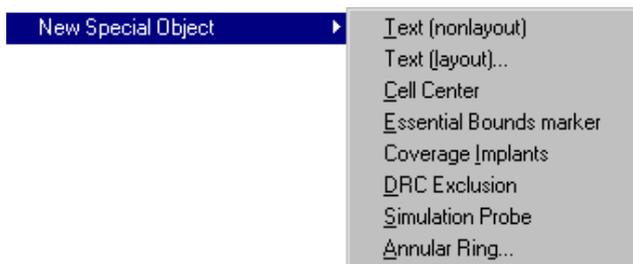


New Pure-Layer Node... [6-10] [7-1]

This command allows the creation of a piece of pure geometry. Pure-layer nodes exist for the purpose of creating unusual layout geometries such as pads or analog circuits. After choosing a node with this command, select a position in the editing window for the node. Initially, pure-layer nodes are square. Other rectangular geometries can be specified with the **Size** command below. Nonrectangular geometries can be created by using the **Outline Edit** command below.



New Special Object



This command allows the creation of certain special components.

Text (nonlayout) [6-8]

is a free-floating piece of text that merely annotates the circuit. It is not a part of fabrication output. When you click, a piece of text is placed (it is initially the word "text" and is highlighted).

Text (layout)... [6-10]

is way to create text that is actual layout. The dialog allows you to specify the text, font, size, scale, and the layer to use (to make the text visible in chip photomicrographs, choose the highest metal layer).

Cell Center [3-3]

is a fiduciary mark that, when placed in a cell, defines the center of that cell. This information is used by the editor when creating or moving instances of that cell. Note that once placed, this object can be selected only by using the *special select* button.

Essential Bounds marker

is a fiduciary mark that, when two or more are placed in a cell, defines the



essential area of that cell. Note that once placed, this object can be selected only by using the *special select* button.

Coverage Implants [7–4]

This command generates pure-layer nodes that cover the implants in the current cell. The nodes are left highlighted so that you can see what has been generated. Previous pure-layer nodes are removed. The new nodes are made "hard to select" to keep them from interfering with normal selection (see [Section 2–1](#)).

DRC Exclusion [9–2]

is a node that marks an area to be ignored by the design-rule checkers (Hierarchical and Dracula). See the **DRC** command in the **Tools** menu for more on the Dracula interface.

Simulation Probe [10–2]

is a node that marks an area to be tested by simulators. See the **DRC** command in the **Tools** menu for more on the Dracula interface.

Annular Ring... [6–10]

presents a dialog for the construction of ring shapes.

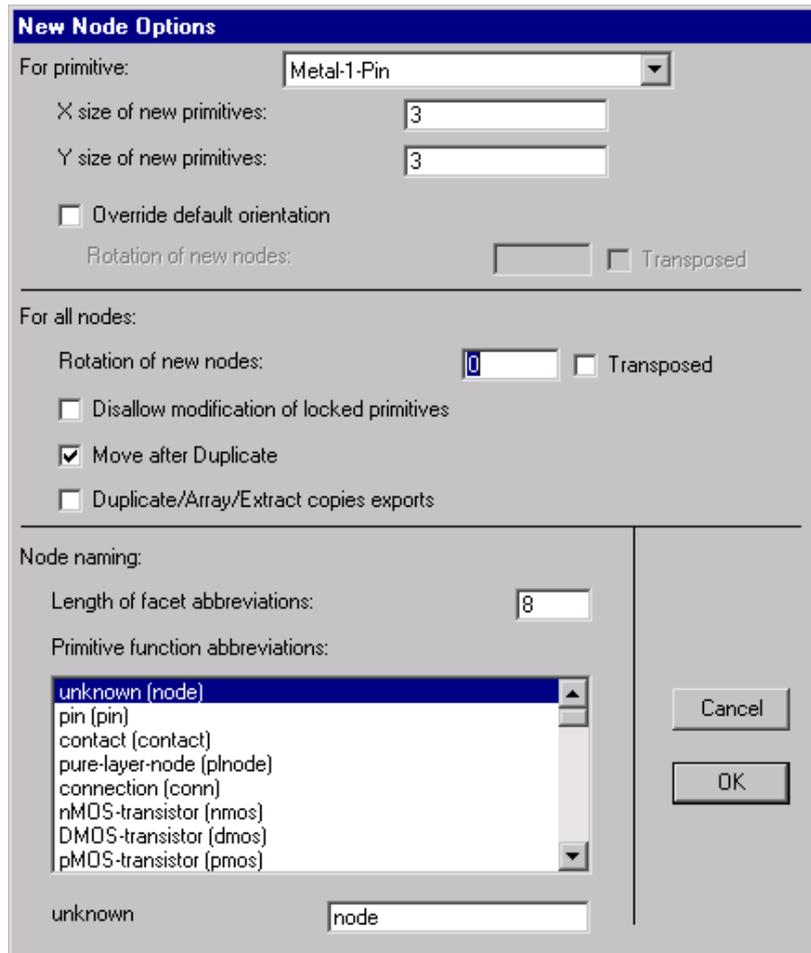
New Node Options... [6–2]

This command presents a dialog that allows default settings to be established for the creation of new nodes. The top section of the dialog lets you set default size and orientation for each primitive node.

The middle section of the dialog contains options that apply to all nodes. The check box "Disallow modification of locked primitives" applies only to primitive node instances in "array" technologies and prevents fixed circuitry from being altered. The check box "Move after Duplicate" allows duplicated objects to be interactively positioned. The check box "Duplicate/Array/Extract copies exports" causes all operations that copy nodes to copy their exports as well. This includes the **Duplicate** and **Array** commands of the **Edit** menu and the **Extract Cell Instance** command of the **Cells** menu. Note that export names must be unique, so the copied exports will have their names modified by either changing array indices or by adding "_1", "_2", etc. to the end. Finally, it is possible to set a default orientation of all new nodes.

The bottom section of the dialog allows you to specify node names to be used for the different types of nodes. These names are used when automatically naming nodes during netlisting.





Cut [6-1] [4-10]

This command copies the currently selected nodes and arcs to the scrap, and then deletes them from the circuit. If you are editing a text window, the selected text is copied and removed. If the Messages window is current, this command copies and removes text from there.

Copy [6-1] [4-10]

This command copies the currently selected nodes and arcs to the scrap. If you are editing a text window, the selected text is copied. If the Messages window is current, this command copies text from there.

Paste [6-1] [4-10]

This command copies the nodes and arcs in the scrap back to the currently selected window. If there are nodes or arcs selected when this command is issued, the copied objects are copied ONTO the selected objects, changing them. If you are editing a text window, text from the text scrap is inserted. If the Messages window is current, this command pastes text there.



Duplicate [6-1]

This command creates a copy of the currently highlighted nodes and arcs. An outline of the duplicated objects attaches to the cursor, and when you click, the objects are placed at that location. If you have disabled "Move after Duplicate" (in the **New Node Options...** command of the **Edit** menu) then the duplicated objects are placed immediately without dragging.

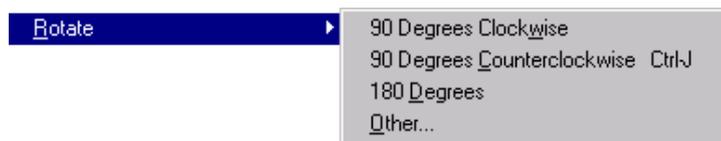
Undo [6-7]

This command reverses the last command made to Electric. This will affect ANY command, not just those that change circuitry. Repeated issuing of this command undoes farther back up to a limit of about 30 commands.

Redo [6-7]

This command redoes changes that were undone by the **Undo** command. Repeated issuing of this command redoes farther up to the last change made.

Rotate [2-6]



This command rotates the currently highlighted component. A submenu allows you to rotate in any of the Manhattan orientations, or provide an arbitrary rotation amount.

Mirror [2-6]



This command flips the currently highlighted components about their horizontal or vertical centerline, according to the submenu.

Size [2-5]

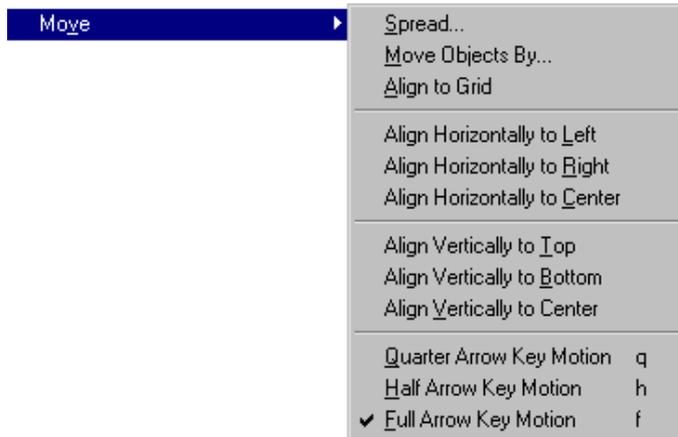


This command alters the size of the currently highlighted components, according to the submenu. The **Interactively** option allows graphical adjustment of the currently selected node or arc (the corner farthest from the cursor is anchored



and the corner closest to the cursor is pulled to the location of the cursor). The **All Selected Nodes** and **All Selected Arcs** options present a dialog for changing the size of the selected nodes or arcs.

Move [2–4] [4–7] [6–5]



The subcommands here affect object movement. You can spread a circuit to make room, move objects by specified amounts, align objects to the grid or to each other, and control the amount of movement that arrow keys use.

Erase [2–3]

This command deletes the currently highlighted objects. Use **Undo** to restore deleted objects. Note that when a node is erased, all connecting arcs and exports are also erased. 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.

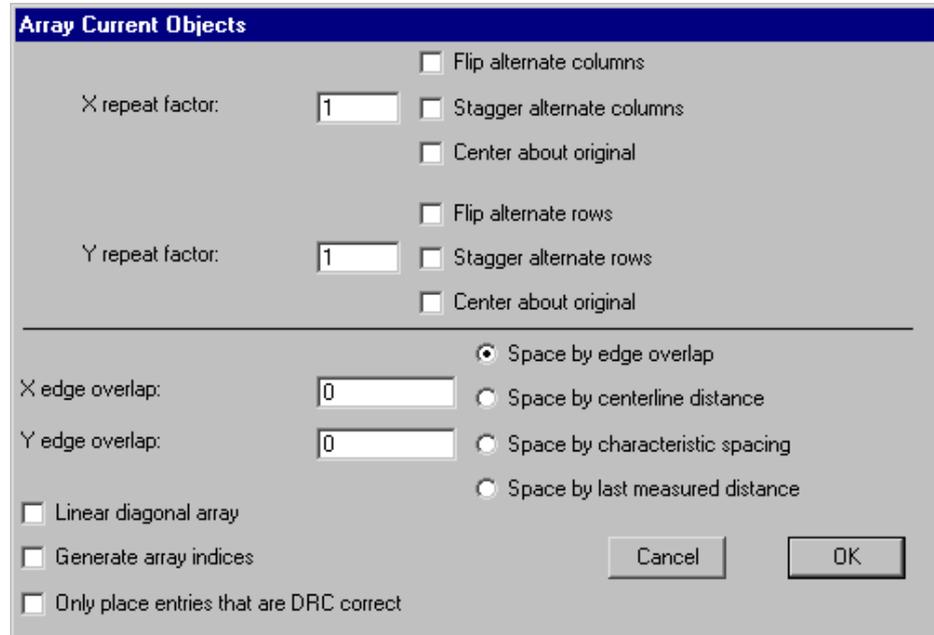
Erase Geometry [2–3]

This command deletes all geometry in the currently highlighted area. Arcs that cross into the area will be truncated at the edge of the area. Note that the area is adjusted by the current alignment values (see [Section 4–7](#)).



Array... [6-4]

This command creates multiple copies of the currently highlighted components. A dialog is presented in which the X and Y repeat factors can be specified. Also, alternate rows and columns can be flipped or staggered, and spacing can be specified in many different ways. If array indices are requested, they appear on each copy.



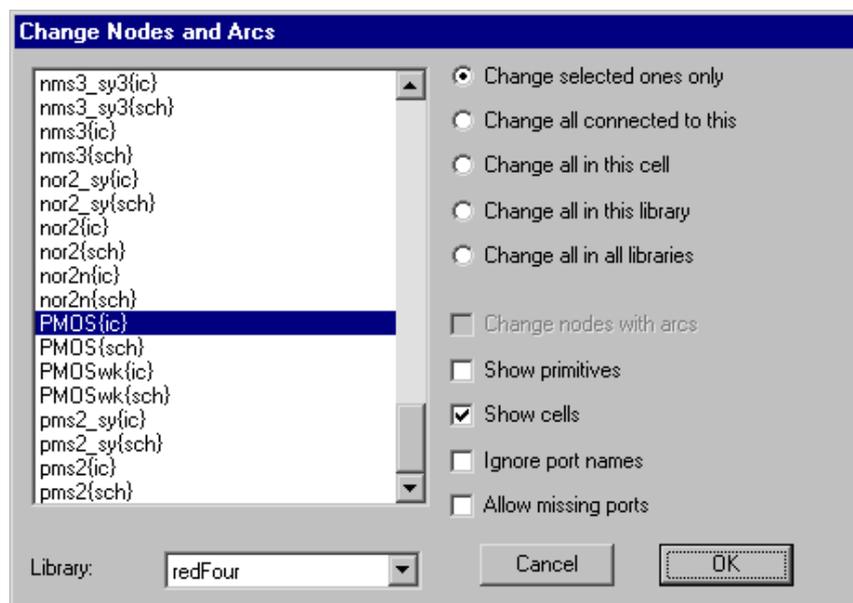
Insert Jog in Arc [2-2]

This command causes the currently selected arc to have a jog inserted. Press the button and move the cursor to see where the jog will be inserted. Release the button to insert at that location. A jog consists of two pin-nodes, one connected to each half of the arc, and another arc connecting them that runs perpendicular to the original arc. Thus, the jog point allows either half of the wire to be moved laterally, while keeping the parts connected. Inserting two jogs allows an arc to form a "U".

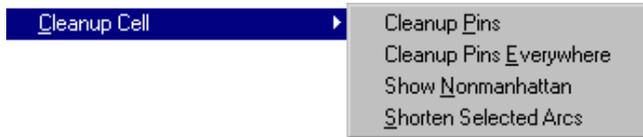
Change... [6-6]

This command causes the selected nodes or arcs to be changed to a different type.

A dialog is presented that allows selection of the new type, the scope of the change, and control over how the change is done.



Cleanup Cell



These commands adjust the current cell so that it is easier to edit.

Cleanup Pins [2-2]

This command examines the current cell and removes isolated and irrelevant pins. It also highlights zero-size nodes.

Cleanup Pins Everywhere [2-2]

This command examines all cells in the current library and removes isolated and irrelevant pins. It also reports zero-size nodes.

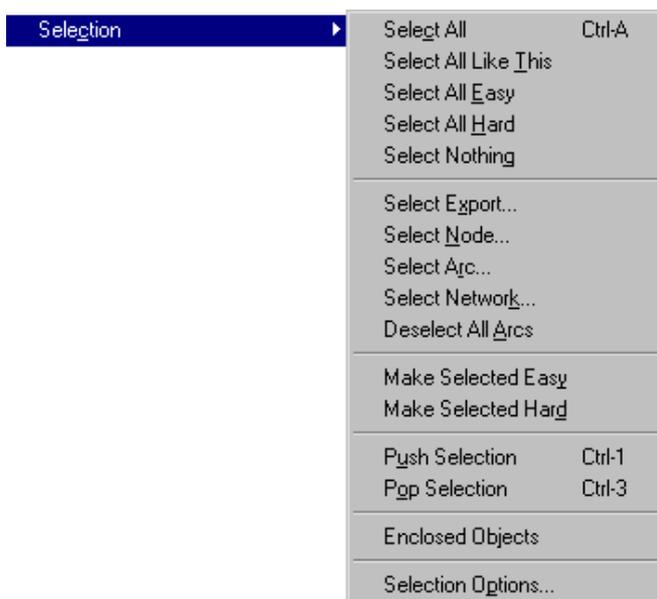
Show Nonmanhattan [5-2]

This command identifies all nonmanhattan geometry in the current cell (nonmanhattan geometry is that which is not at right angles with the coordinate axes). In addition, it tells you of other nonmanhattan geometry elsewhere in the database.

Shorten Selected Arcs [5-2]

This command reduces the length of all selected arcs to their minimum, without moving the attached nodes. This works only on arcs that are connected to large ports, and have room to move inside of the port.

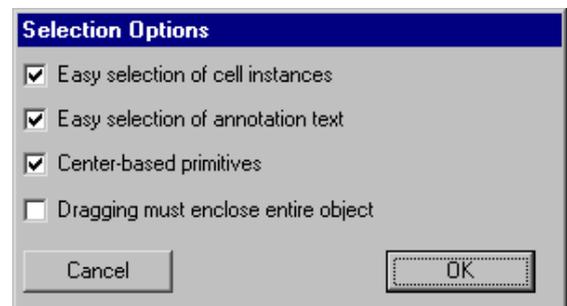
Selection



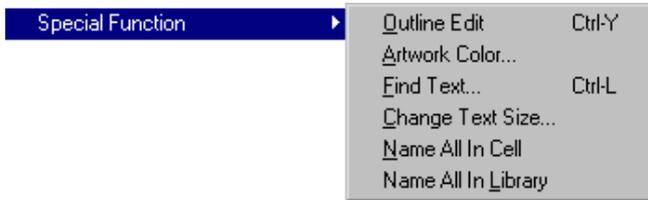
These commands control selection of objects.



Select All [2-1]	This command highlights every node and arc in the current cell.
Select All Like This [2-1]	This command highlights every node and arc in the current cell that is the same as the currently selected node or arc.
Select All Easy [2-1]	This command highlights every node and arc in the current cell that is marked as easy-to-select.
Select All Hard [2-1]	This command highlights every node and arc in the current cell that is not marked easy-to-select.
Select Nothing [2-1]	This command removes all selection in the current cell.
<hr/>	
Select Export... [3-6]	This command presents a list of ports in the current cell and highlights one.
Select Node... [2-4]	This command presents a list of nodes in the current cell and highlights one.
Select Arc... [2-4]	This command presents a list of arcs in the current cell and highlights one.
Select Network... [6-9]	This command presents a list of networks in the current cell and highlights one.
Deselect All Arcs [2-1]	This command causes all selected arcs to be deselected.
<hr/>	
Make Selected Easy [2-1]	This command changes all selected objects to be easy-to-select.
Make Selected Hard [2-1]	This command changes all selected objects so that they are not easy-to-select.
<hr/>	
Push Selection [2-1]	This command saves the currently selected objects on a stack.
Pop Selection [2-1]	This command restores the previously pushed objects on a stack.
<hr/>	
Enclosed Objects [2-1]	This command changes an area selection into an actual selection of nodes and arcs. If you have used the <i>rectangle select</i> button to define an arbitrary rectangular area on the screen, this command replaces that selection with the actual nodes and arcs in that area.
<hr/>	
Selection Options... [2-1]	This command controls selection options, including the setting of "hard-to-select" status, the use of node centers instead of their lower-left corner, and how area selection is done.



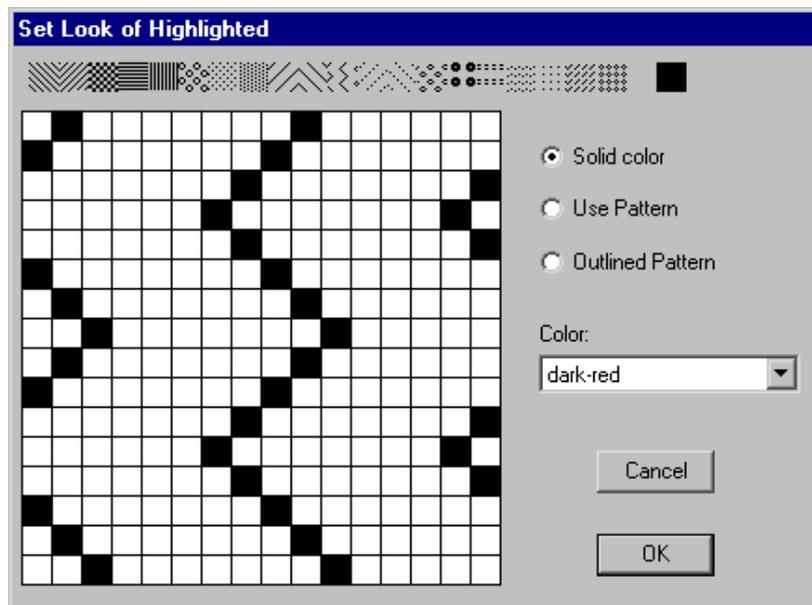
Special Function



These commands offers special operations on specific objects.

Outline Edit [6-10] This command allows the currently highlighted node to have its outline manipulated. It works only on certain nodes for which the outline information has meaning. MOS transistors use this information as the centerline of the gate in a serpentine description. Pure-layer nodes use this information to describe their shape. Finally, some primitives from the Artwork technology can use this information to describe their shape. Once this command is issued, the buttons are redefined to apply specifically to outline editing. The *selection* button selects and moves a point on the outline, and the *creation* button creates a new point. Also, the **Erase**, **Rotate**, and **Mirror** commands of this menu change meaning when editing outlines. Use the **Get Outline Info** command of the **Info** menu to see outline coordinates. To terminate outline editing, this menu entry changes to **Exit Outline Edit** which, when reissued, restores the commands and mouse buttons.

Artwork Color... [7-7] This command causes the highlighted node or arc to be given a specific color and/or pattern. You can set individual bits in the pattern or choose from a set of predefined patterns along the top.

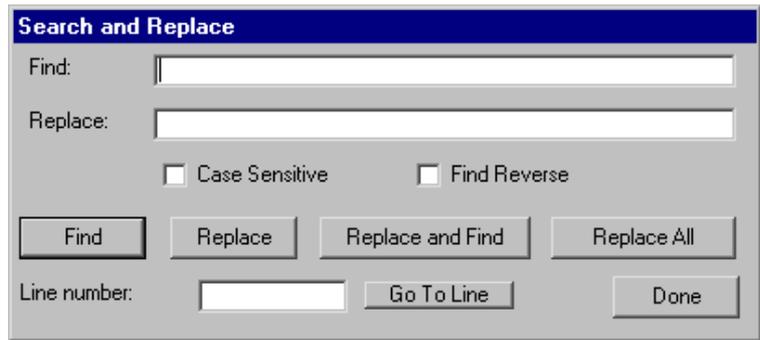


The object must be a primitive from the Artwork technology for this command to work. Another way to examine the color of the currently highlighted node or arc is to use the **Get Info** command of the **Info** menu.



Find Text...
[4–10]

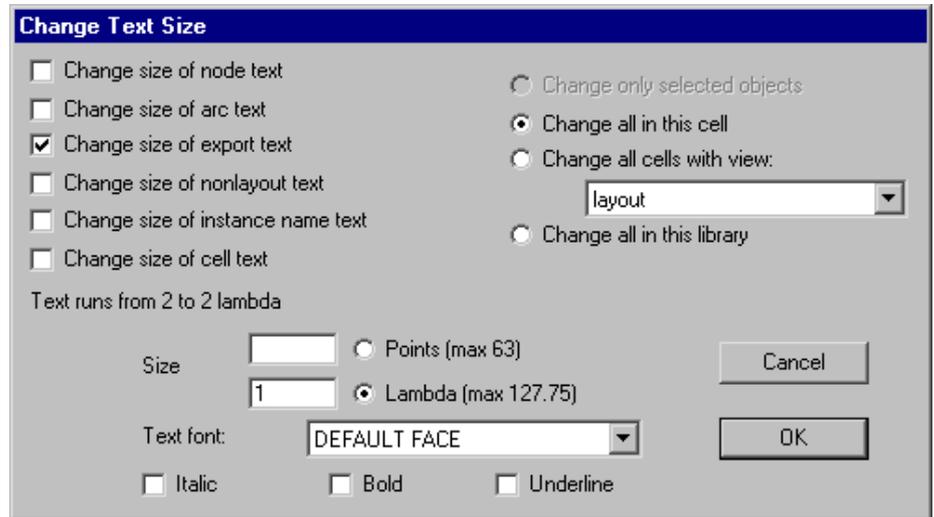
This command presents a dialog for search and replacement of text. It works both in text cells and in circuits.



Change Text Size...
[6–8]

This command presents a dialog for changing the size of text objects.

You can change as many different categories of text as you like, and can change those types of text locally or globally.



Name All In Cell
[2–4]

This command names every node and arc in the current cell. The names do not appear, but are present for netlisters and debugging. (Note that netlisters that need names do this automatically.)

Name All In Library
[2–4]

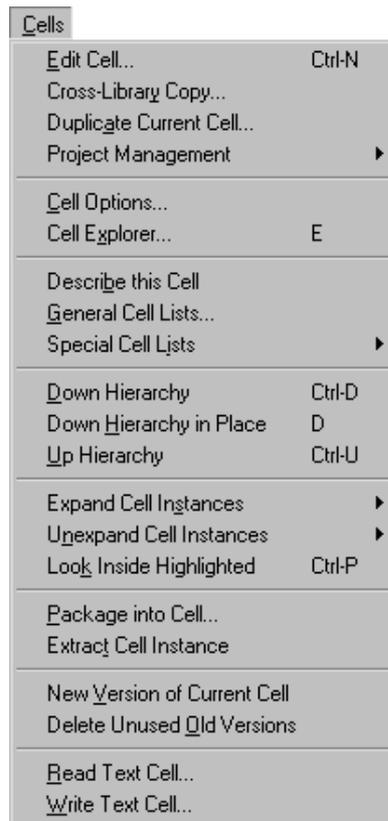
This command names every node and arc in the current library. The names do not appear, but are present for netlisters and debugging. (Note that netlisters that need names do this automatically.)



Chapter 12: MENU SUMMARY



12-3: The Cells Menu

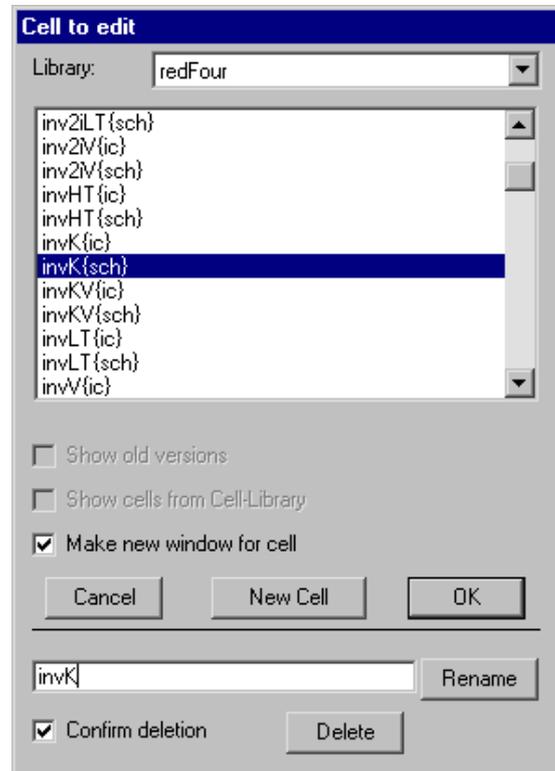


Cell cells are collections of circuitry. Each cell of a cell has a view type (layout, schematic, etc.) and a version number. Because cell instances may be placed inside of other cells, the cell notion defines hierarchy in Electric.



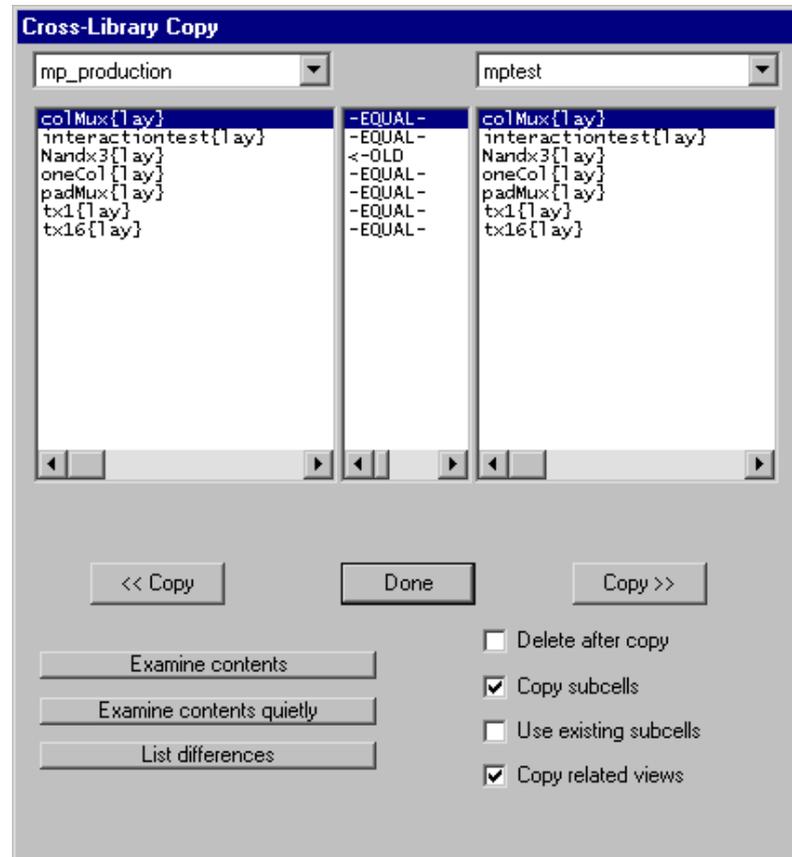
Edit Cell... [3-2]

This command displays an existing cell or creates a new cell in the current window. You are prompted for its name and view type. Use the "New Cell" button to create a new one.



Cross-Library Copy... [3-10]

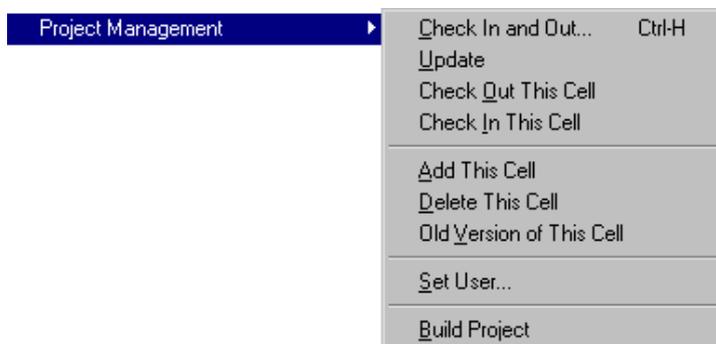
This command presents a dialog that shows cells in two different libraries, and indicating which ones are newer. You can select a cell in either library and copy or move it to the other. In addition, if the cell has subcells or associated views that are newer, they can also be copied. The "Examine Contents" buttons compare the contents of cells and displays an indication of whether they are actually different or just out of date.



Duplicate Current Cell [3-2]

This command makes a copy of the cell in the current window and gives it a new name. You will be prompted for the new name. The new cell is placed in the same library as the old one.

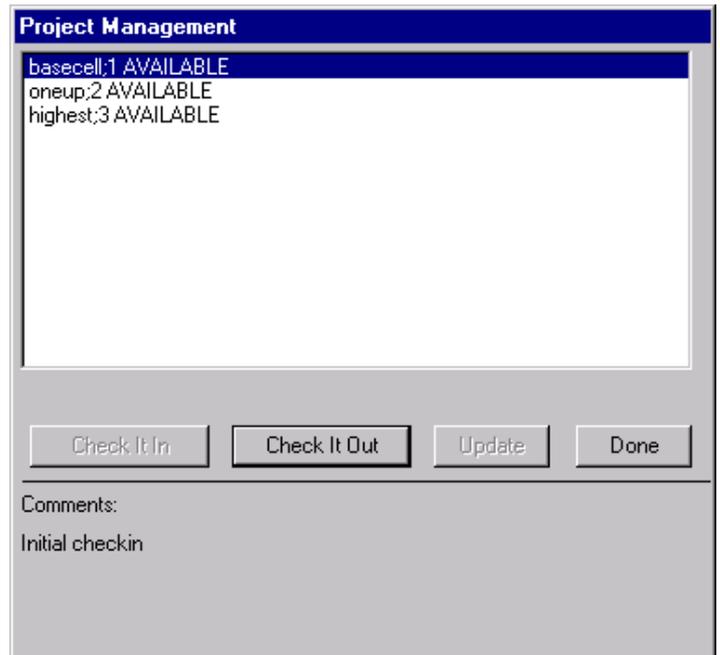
Project Management [6-11]



This set of commands controls the interactions of multiple users working together on a single library. Subcommands are available for checking cells in and out, for updating your personal copy of the library from a master library, and other features.



Check In and Out... Presents a dialog showing all cells in the library, and allowing you to check them in and out of the project management system.



Update Updates the current library from the master library.

Check Out This Cell Checks out a cell from the master library.

Check In This Cell Checks the current cell into the master library.

Add This Cell Adds the current cell to the master library (if it is not part of the project management system).

Delete This Cell Deletes a cell from the master library.

Old Version of This Cell Access old versions of a cell.

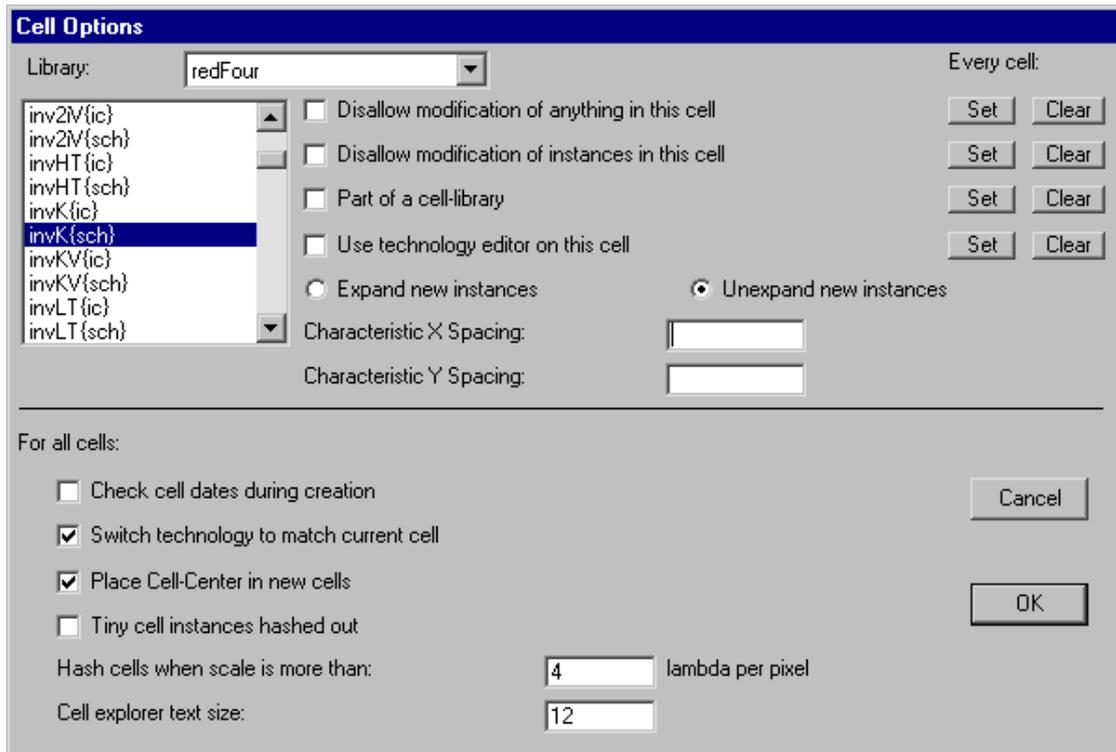
Set User... Sets user identification for project management.

Build Project Converts the current library into a master library for subsequent sharing.

Cell Options... [3-7]

This command presents a dialog that allows cell options to be set. There are check boxes at the top for controlling cell modifications, recognizing cell libraries, and recognizing technology editor libraries. Buttons on the right can set or clear these bits for all cells. New instances can be expanded or unexpanded. You can also set each cell's "Characteristic Spacing" (used when arraying, see [Section 6-4](#)).





The bottom half of the dialog has options that apply to all cells. The check box "Check cell dates during creation" causes creation and modification information to be used to ensure that the hierarchy has been built in the proper order. The check box "Switch technology to match current cell" causes the current technology to change whenever the current cell changes so that it is appropriate to that cell. The check box "Place Cell-Center in new cells" requests that all newly created cells have a Cell-Center node (named "Facet-Center" for historical reasons) placed at the origin (see [Section 3-3](#) for more on Cell centers). The check box "Tiny cell instances hashed out" causes the contents of cell instances to be a gray hash area when zoomed too far out to be distinguishable. You can control the number of lambda per pixel that triggers the hashing-out of cells.

Cell Explorer... [\[3-7\]](#)

This command splits the current window, and shows a hierarchical "explorer" window in the left half. A recursively indented list of cells-within-cells is listed, and you can explore your circuit's hierarchy. Note that the cell explorer can also be invoked by clicking on the "tree" icon in the lower-left corner of the window.

Describe this Cell [\[3-7\]](#)

This command displays information about the cell in the current window.



General Cell Lists... [3-7]

These presents a dialog for selecting a subset of the cells.

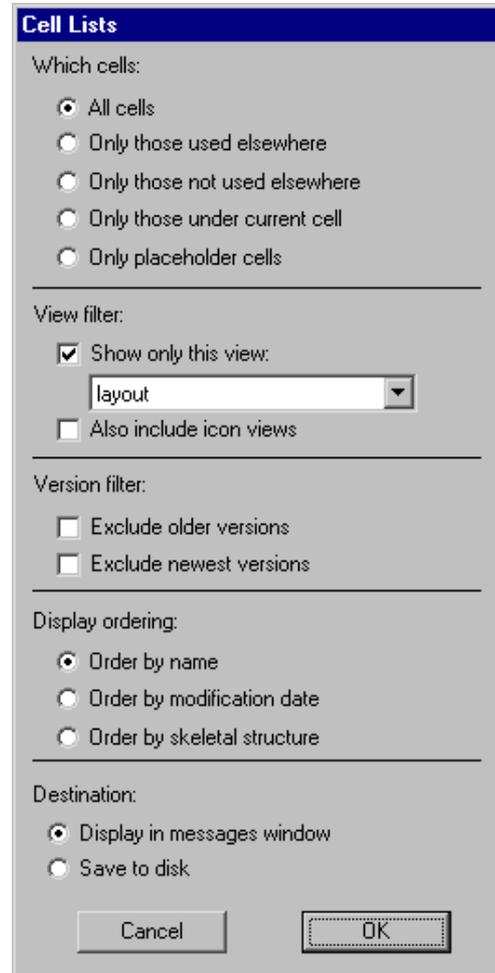
The section labeled "Which cells:" selects the cells to be listed (all, only those used/not used in other cells, only those in the current cell, or only "placeholder" cells: those created because of cross-library dependency failures, see Section 3-9).

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.



Special Cell Lists [3-7]



List Nodes in this Cell

This set of commands give specialized information about cells.

This command displays a list of all nodes in the current cell.

List Cell Instances

This command displays a list of cell instances contained in the current cell.

List Cell Usage...

This command prompts for a cell and displays a list of places where the it exists as an instance.

Graphically, Entire Library

Creates a graph of every cell in the library. The graph is actually a new



cell that contains Artwork components. Thus, it will be stored with the library and may be edited, printed, etc.

Graphically, From Current Cell

Creates a graph showing the current cell at the top, and all subcells below it. The graph is actually a new cell that contains Artwork components. Thus, it will be stored with the library and may be edited, printed, etc.

Down Hierarchy [3-5]

This command changes the current edit window so that the cell whose instance is currently highlighted is now the editable cell. When an icon is selected, its contents (schematic) is edited. If the icon is inside of its own schematic (a documentation feature) then going down does edit the icon.

Down Hierarchy in Place [3-5]

This command changes the current edit window so that the cell whose instance is currently highlighted is now the editable cell. However, if the instance is transformed (rotated or mirrored), it is edited in that orientation.

Up Hierarchy [3-5]

This command returns editing to the higher level of hierarchy in which the current cell is instantiated. It thus travels back up the hierarchy that was descended with the **Down Hierarchy** command. If an export is highlighted, that network is highlighted in the outer cell. If there was no **Down Hierarchy**, and the higher level cell cannot be determined, a list of possible cells will be presented. Arbitrary depth of hierarchy can be traversed with these two commands by repeating them.

Expand Cell Instances [3-4]



These commands cause the highlighted cell instances to be *expanded*, which means that their contents will be displayed.

If their contents is already being displayed, this will cause the contents of any subcells to be displayed, repeatedly down the hierarchy. Once expanded, these cell instances will always display their contents until the **Unexpand Cell Instances** commands are issued. For a temporary view of the contents of an instance, use **Look Inside Highlighted**. Note that the expansion information can also be set in the **Get Info** dialog of the **Info** menu.

One Level Down



All the Way

Specified Amount...

The next level of unexpanded cells is made visible.

All cells from here to the bottom of the hierarchy are made visible.

You are prompted for a number of levels of hierarchy, and that many levels of depth are made visible.

Unexpand Cell Instances [3–4]



These commands cause the highlighted cell instances to be *unexpanded*, which means that they will be displayed as black boxes.

If there are expanded subcells, these are closed first and subsequent commands will unexpand up the hierarchy. Note that the expansion information can also be set in the **Get Info** dialog of the **Info** menu.

One Level Up

The next level of expanded cells, from the bottom of the hierarchy, is unexpanded.

All the Way

All cells from here to the bottom of the hierarchy are closed.

Specified Amount...

You are prompted for a number of levels of hierarchy, and that many levels of depth are closed.

Look Inside Highlighted [3–4]

This command displays all layout in the currently highlighted area, all the way down the hierarchy. This "peek" into the cell instances is temporary and will not be shown again if the window is redrawn in any way. To get a permanent view into a cell, use the **Expand Cell Instances** command above or set the "Expand" option in the **Get Info** command of the **Info** menu.

Package into Cell... [3–8]

This command creates a new cell, possibly in a new cell, that contains all of the circuitry in the currently highlighted area. You will be prompted for the new cell name. The highlighted area is defined as the bounding rectangle of everything that is highlighted. Arcs that cross outside of the bounds will not be copied. A more precise way of defining a highlighted area is to use the *rectangle select* button.

Extract Cell Instance [3–8]

This command takes the currently highlighted cell instance and replaces it with its contents. This is conceptually the opposite of **Package into Cell...** as it removes a level of hierarchy. If multiple cell instances are highlighted, all of them are extracted.



New Version of Current Cell [3–2]

This command makes a copy of the cell in the current window. Version numbers appear in the cell name as a semicolon followed by a number (except for the most recent cell which has no version notation). For example, after the duplication of cell MyCircuit, there will be the cells MyCircuit and MyCircuit;1. The current cell, which has no explicit version number in its name, actually is version 2 (smaller version numbers are older).

Delete Unused Old Versions [3–2]

This command deletes all cells in the current library that are not the most recent version and are not being used as instances in some other cell. It essentially cleans up the library when too many versions have been created. As with all commands, this can be undone with the **Undo** command of the **Edit** menu.

Read Text Cell... [4–10]

This command reads a disk file into the current text window. It replaces the existing contents.

Write Text Cell... [4–10]

This command saves the current text window to disk.

 [Previous](#)

 [Table of Contents](#)

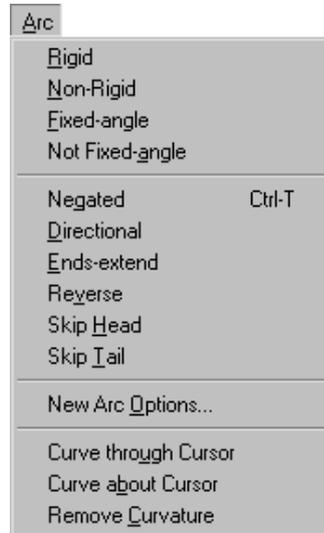
[Next](#) 



Chapter 12: MENU SUMMARY



12-4: The Arc Menu



Arcs are the wires that connect components. Once they have been created, they may be modified to indicate constraints (rigid or fixed-angle), they may hold negating bubbles (only in schematics) and have other properties.

Rigid [5-3]

This command causes the currently highlighted arcs to become rigid, which means that they remain at a fixed length and orientation relative to their connecting nodes. Rigid arcs display the letter "R" when highlighted.

Non-Rigid [5-3]

This command causes the currently highlighted arcs to become nonrigid, which means that they can stretch with their connecting nodes.

Fixed-angle [5-3]

This command causes the currently highlighted arcs to become angle-fixed, which means that they remain at a constant angle through all node changes. Thus, if a node on one end moves, the other end's node may also move to keep the arc from changing orientation. Fixed-angle arcs display the letter "F" when highlighted.



Not Fixed–angle [5–3]

This command causes the currently highlighted arcs to become non–fixed–angle, which means that they can slant arbitrarily to connect their two nodes.

Negated [5–4]

This command causes the currently highlighted arcs to have a bubble on one end. If there is already a bubble, it is removed. This command works only for wires in schematics.

Directional [5–4]

This command causes the currently highlighted arcs to have a directional arrow on one end. If there is already a directional arrow, it is removed. This arrow is for documentation purposes only.

Ends–extend [5–4]

This command causes the currently highlighted arcs to stop extending by half of their width. If they are already not extended, they revert to extending by half of their width.

Reverse [5–4]

If the negating bubble or the directional arrow is on the wrong end of an arc, this command reverses the location.

Skip Head [5–4]

This command causes special directives to the head of the arc to be ignored. If the arc is directional (with an arrow at the head), then the arrow is not drawn. If the arc has had end–extension turned off, then the head of the arc is end–extended.

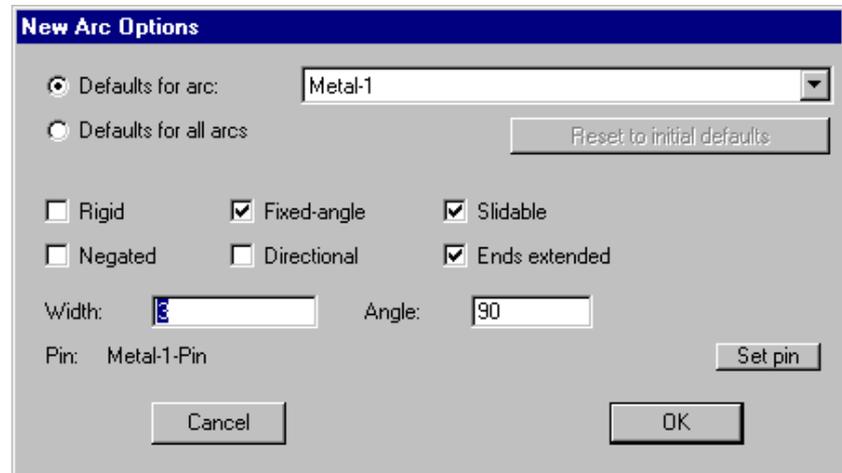
Skip Tail [5–4]

This command causes special directives to the tail of the arc to be ignored. If the arc is negated (with a bubble at the tail), then the bubble is not drawn. If the arc has had end–extension turned off, then the tail of the arc is end–extended.



New Arc Options... [5-5]

This command presents a dialog that allows default settings to be established for the creation of new arcs. Specific arc types may be given default constraint settings, default widths, and default angle increments. It is also possible to set an overriding constraint set for all new arcs.



Curve through Cursor [5-4]

This command causes the currently highlighted arc to be curved, such that it passes through the location of the cursor. After issuing this command, click on the screen to specify the point through which the arc will pass. This command works only for arcs in the RCMOS and Artwork technologies.

Curve about Cursor [5-4]

This command causes the currently highlighted arc to be curved, such that it wraps around the location of the cursor. After issuing this command, click on the screen to specify the point which will be the center of the arc's curvature. This command works only for arcs in the RCMOS and Artwork technologies.

Remove Curvature [5-4]

This command causes the currently highlighted curved arc to be straight again. This command works only for arcs in the RCMOS and Artwork technologies.

 [Previous](#)

 [Table of Contents](#)

[Next](#) 



Chapter 12: MENU SUMMARY



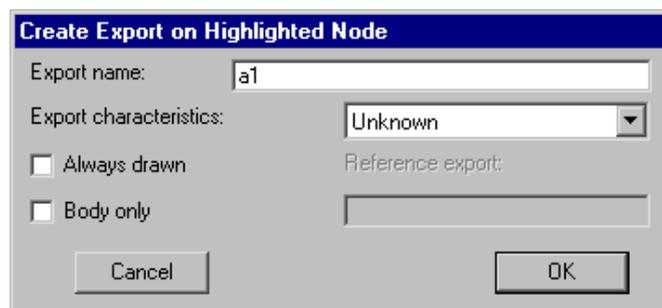
12-5: The Export Menu



Ports are the sites of arc connections on nodes. Primitive nodes have ports automatically defined, but cell instances (complex nodes) have no ports on them. The locations of arc connections on these instances must be defined by creating *exports* inside of the cell's definition. These are simply ports that have been "exported".

Create Export... [3-6]

This command takes the currently highlighted port on the currently highlighted node and makes it an export of the current cell. The export then becomes a port on instances of the cell, higher up the hierarchy.



A dialog will be presented for the name and characteristics of the export. The characteristics may be



directional (input, output, bidirectional), levels (power or ground), clocking (clock with optional phases 1 through 6), or reference (with an associated reference export name).

The option "Always drawn" indicates that the export name will not be suppressed when the port is connected or otherwise in use. The option "Body only" indicates that this export will not be included in an iconic view of the cell.

Re-Export Everything [3-6]

This command automatically creates exports in the current cell, wherever a port on a sub-cell instance is found to be unconnected and unexported. All such ports on subcell instances are exported. This is useful in array-based design where the edges of the array, which are not connected, should be exported further up the hierarchy.

Re-Export Highlighted [3-6]

This command does the same thing as **Re-Export Everything**, except that it functions only on the currently highlighted nodes.

Re-Export Power and Ground [3-6]

This command does the same thing as **Re-Export Everything**, except that it functions only on Power and Ground exports.

Delete Export [3-6]

This command removes the export on the currently highlighted node.

Delete All Exports on Highlighted [3-6]

This command removes the all exports on all of the currently highlighted nodes.

Delete All Exports in Area [3-6]

This command removes the all exports in the selected area.

Move Export [3-6]

This command moves an export from one node to another. The source port is selected in the standard fashion and the destination port is specified with the *toggle select* button. Be sure that the correct port is highlighted on both the source and destination nodes.



Rename Export... [3-6]

This command allows you to rename any of the exports in the current cell.

Summarize Exports [3-6]

This command makes a condensed summary of the exports in the current cell.

List Exports [3-6]

This command lists the exports in the current cell.

Show Exports [3-6]

This command shows all exports in the current cell (drawing a line from the export to the edge of the display). The display of export location is temporary, and goes away when the cell is redrawn.

Port and Export Options... [3-6]

This command affects the way that ports and exports are drawn in the current window. "Full Port Names" causes the complete name to be drawn. "Short Port Names" displays port names up to the first nonalphabetic character. For example, the port names "In" and "In.17" will both display as "In". "Ports as Crosses" causes "+" signs to be drawn, instead of text. To remove port display completely, use the **Layer Visibility** command of the **Windows** menu.



Show Ports on Node [3-6]

This command shows all port locations on the currently selected nodes. The display of port location is temporary, and goes away when the cell is redrawn.



Add Exports from Library... [3-6]

This command copies exports from one library to another. It prompts for another library and locates exports in that library that should be in the current one. It does this by finding cells in the other library that have the same name as those in the current library. New exports are then created in the current library to match the location of those in the other library.

This command is useful in managing standard cell libraries that are imported from other file formats. Because some formats contain geometry and others contain connectivity, this command is needed to put them together.

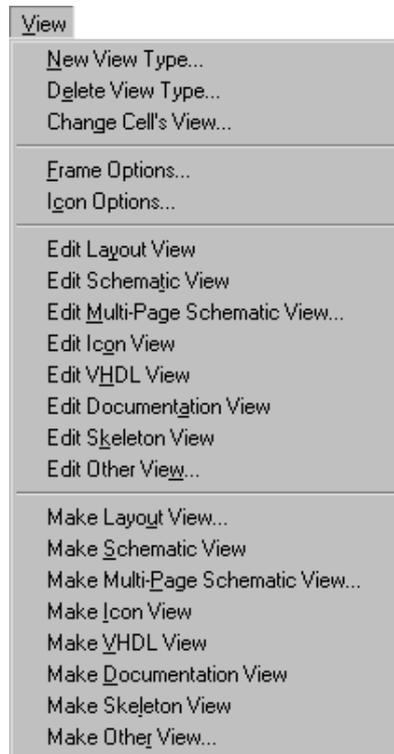
[← Previous](#) [↑ Table of Contents](#) [Next →](#)



Chapter 12: MENU SUMMARY



12-6: The View Menu

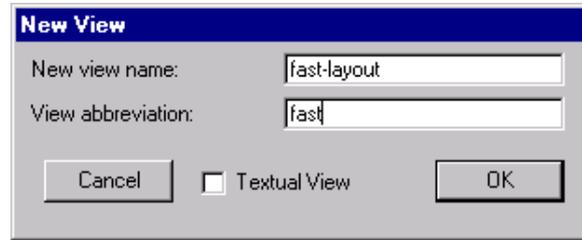


Each cell has a view associated with it. Although Electric defines a standard set of view types (layout, schematic, icon, skeleton, simulation-output, VHDL, document, and many netlist formats) it is possible to define new view types (for example, "fast layout") and to assign these views to cells. Special commands switch between the different views of a cell. This menu also controls the display of schematic frames. Finally, there are commands that automatically generate new cells of a different view type.



New View Type... [3-11]

This command creates a new type of view for cells. You will be prompted for the name of the view and an abbreviation to use in cell names. The abbreviation appears in curly brackets after the cell name, for example MyAdder{fast}.



Delete View Type... [3-11]

This command deletes a view type. There cannot be any existing cells with this view. Also, it is not possible to delete the basic view types that are defined by Electric.

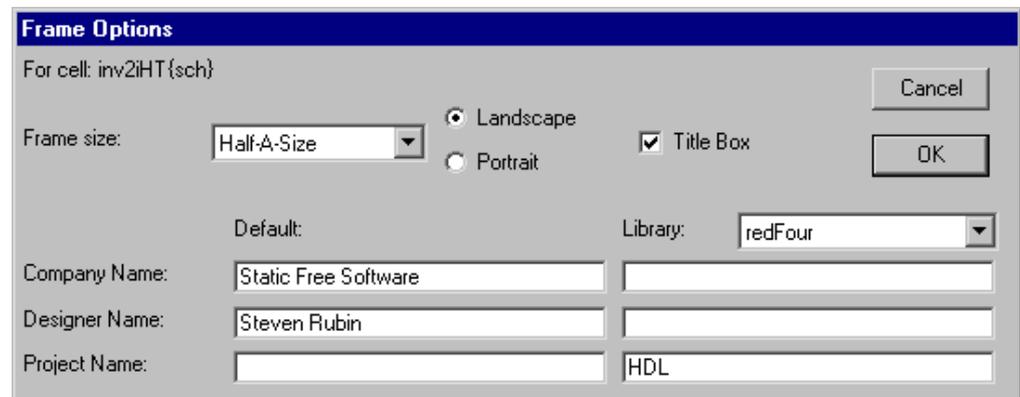
Change Cell's View... [3-11]

This command changes the view type of the current cell. The cell name remains the same, but the view changes. You will be prompted for the new view. Note that this is one of the few commands in Electric that is NOT undoable.

Frame Options... [7-6]

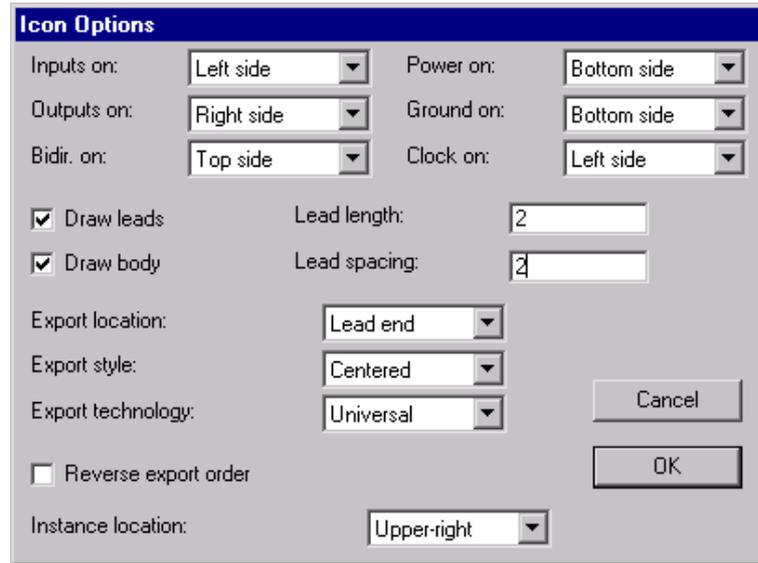
This command allows you to control the drawing of a frame in the current cell. You can choose among 6 frame sizes ("Half-A", "A", "B", "C", "D", or "E") and also whether they are wider (landscape) or taller (portrait).

In addition, you can provide information that will be displayed in the corner of the frame (currently, only the company name, project name, and designer name).



Icon Options... [3-12]

This dialog controls how icons are created from schematics. For each type of port in the schematic, you can control on which of the four sides of the icon the port will appear. You can also control the location and order of ports and the style of their text. You can control whether the icon has a body or leads for each port, the size and spacing of the leads, and which technology to use for the export connections. Finally, you can control where the newly created icon's instance will appear in the schematic.



Edit Layout View [3-11]

If the current cell has an associated layout view, a new window is created to display that view.

Edit Schematic View [3-11]

If the current cell has an associated schematic view, a new window is created to display that view.

Edit Multi-Page Schematic View... [3-11]

If the current cell has an associated multi-page schematic view, a new window is created to display that view. You will be prompted for the page number. Multi-page schematics are described with the "pN" view type, where "N" is the page number. For example, to edit page 6 of cell "happy", look for the cell called "happy{p6}".

Edit Icon View [3-11]

If the current cell has an associated icon view, a new window is created to display that view.

Edit VHDL View [3-11]

If the current cell has an associated VHDL view, a new window is created to display that view. Note that the VHDL view is a textual view, so you will now be using a text editor. See [Section 4-10](#) for more on text editing.



Edit Documentation View [3-11]

If the current cell has an associated documentation view, a new window is created to display that view. Note that the documentation view is a textual view, so you will now be using a text editor. See [Section 4-10](#) for more on text editing.

Edit Skeleton View [3-11]

If the current cell has an associated skeleton view, a new window is created to display that view.

Edit Other View... [3-11]

If the current cell has an associated other view, a new window is created to display that view. You will be prompted for the view type that you wish to edit.

Make Layout View... [3-12]

This command creates a new layout cell of the cell in the current editing window (or a new version of the layout view if one already exists). It is presumed that the current cell is a layout or schematic view and that the new cell is layout in a different but similar technology. You will be prompted for the new technology.

Make Schematic View [3-12]

This command creates a new schematic cell of the cell in the current editing window. It is presumed that the current cell is a layout view. If there is already a schematic view of this cell, a new version of the schematic is created.

Make Multi-Page Schematic View... [3-11]

This command creates a new page of a multi-page schematic cell of the cell in the current editing window. You will be prompted for the page number. Multi-page schematics are described with the "pN" view type, where "N" is the page number. For example, to edit page 6 of cell "happy", look for the cell called "happy{p6}". If there is already a multi-page schematic view of this cell with the requested page number, a new version of that page's cell is created.

Make Icon View [3-12]

This command creates a new icon cell of the cell in the current editing window. The new cell has the same ports but a black-box look with input ports on the left, outputs on the right, power and ground on the top, and clock lines on the bottom. The **Icon Options** command controls the generation of icons.

Make VHDL View [3-12]

This command creates a new VHDL cell (a textual view) of the cell in the current editing window. All subcells of the current cell are also translated to VHDL, and their descriptions are stored in views of the subcell. If "VHDL stored in cell" is unchecked (in the **VHDL Options...** subcommand of the **VHDL Compiler** command of the **Tools** menu), all VHDL will be written to disk.



Make Documentation View [3-11]

This command creates a new Documentation cell (a textual view) of the cell in the current editing window.

Make Skeleton View [3-12]

This command creates a new skeleton cell of the cell in the current editing window. The new cell has the same size and ports, but none of the internal layout.

Make Other View... [3-11]

This command creates a new cell of the cell in the current editing window. You will be prompted for the view type of the new cell.

 [Previous](#)

 [Table of Contents](#)

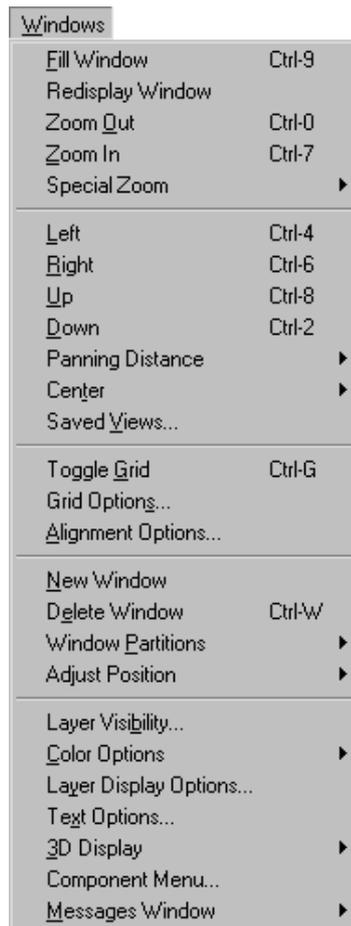
[Next](#) 



Chapter 12: MENU SUMMARY



12-7: The Windows Menu



These commands allow manipulation of the editing window. Arbitrary zooming and panning may be done. A grid can be displayed. The window can be split multiple times to show different cells or even different areas of the same cell. Colors, layers, and port labels can be manipulated. Even the menu of components on the left can be altered.

Fill Window [4-4]

This command causes the cell in the current window to be shifted and scaled so that it fits inside of the window.



Redisplay Window [4-1]

This command causes the cell in the current window to be redrawn.

Zoom Out [4-4]

This command causes the cell in the current window to be halved in size so that more of it appears.

Zoom In [4-4]

This command causes the cell in the current window to be doubled in size so that only the center area remains visible.

Special Zoom [4-4]



These commands change the magnification of the display in special ways.

Focus on Highlighted

This command shifts and scales the cell in the current window so that the currently highlighted area fills the window. The highlighted area is defined as the bounding rectangle of everything that is highlighted. A more precise way of defining a highlighted area is to use the *rectangle select* button to drag a rectangle on the screen.

Highlight then Focus

This command waits for the user to drag an area on the screen. After the area has been identified, the window is zoomed so that the area fills the screen.

Make Grid Just Visible

This command zooms to the point where the grid is visible, but any further zoom-out would make it invisible. The grid is displayed.

Match Other Window

This command scales the cell in the current window so that its size matches that of the other window. If there are more than two windows, you will be prompted for the window to match.



Left [4-4]

This command causes the cell in the current window to shift left (the window actually moves right). Use the subcommands of the **Panning Distance** command to control the amount of the shift.

Right [4-4]

This command causes the cell in the current window to shift right (the window actually moves left). Use the subcommands of the **Panning Distance** command to control the amount of the shift.

Up [4-4]

This command causes the cell in the current window to shift up (the window actually moves down). Use the subcommands of the **Panning Distance** command to control the amount of the shift.

Down [4-4]

This command causes the cell in the current window to shift down (the window actually moves up). Use the subcommands of the **Panning Distance** command to control the amount of the shift.

Panning Distance [4-4]



These commands change the amount by which the panning commands move.

Small

This command causes panning commands to shift the screen by 0.15 of its size.

Medium

This command causes panning commands to shift the screen by 0.3 of its size.

Large

This command causes panning commands to shift the screen by 0.6 of its size.

Center [4-4]



These commands causes the cell in the current window to be shifted so that the desired point is in the center of the window.

Selection

This command causes the cell in the current window to be shifted so that the currently highlighted objects are in the center of the window.

Cursor



This command causes the cell in the current window to be shifted so that the current cursor location is in the center of the window.

Saved Views... [4-4]

This command presents a dialog for saving and retrieving window views (a zoom and pan amount). Saved views are given names which can be used to restore them later.

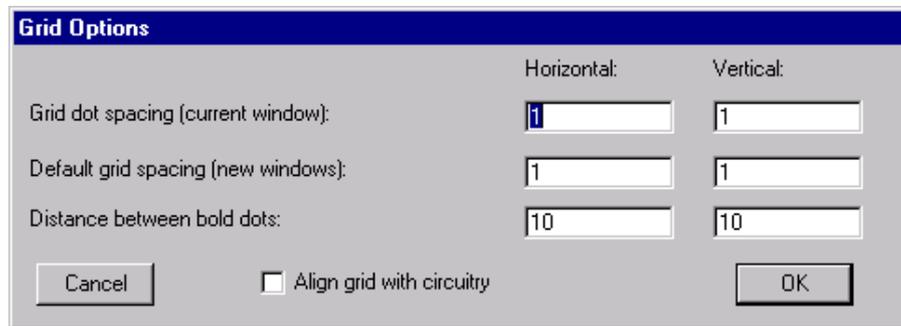
Toggle Grid [4-7]

This command turns on or off the display of the grid.

Grid Options... [4-7]

This command presents a dialog that allows control of the grid spacing in the current window.

It also allows you to set the default grid spacing for new windows, the number of grid dots between bold ones, and whether or not the grid is aligned with the circuitry.

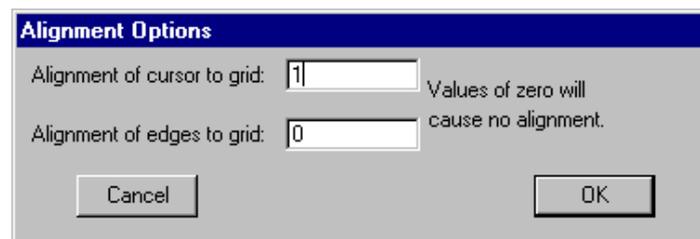


The screenshot shows the 'Grid Options' dialog box. It has a title bar 'Grid Options' and a blue header. The dialog contains three rows of input fields: 'Grid dot spacing (current window):' with a value of '1', 'Default grid spacing (new windows):' with a value of '1', and 'Distance between bold dots:' with a value of '10'. To the right of these fields are two columns labeled 'Horizontal:' and 'Vertical:', each with a corresponding input field containing '1'. At the bottom, there is a 'Cancel' button, a checkbox labeled 'Align grid with circuitry' which is unchecked, and an 'OK' button.

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

Alignment Options... [4-7]

This command presents a dialog that allows control of the alignment of the cursor to grid points, and the alignment of object edges to grid points. Note that the current alignment of the cursor to grid points is shown in the status area under the heading "ALIGN".



The screenshot shows the 'Alignment Options' dialog box. It has a title bar 'Alignment Options' and a blue header. The dialog contains two rows of input fields: 'Alignment of cursor to grid:' with a value of '1' and 'Alignment of edges to grid:' with a value of '0'. To the right of these fields is a note: 'Values of zero will cause no alignment.' At the bottom, there are 'Cancel' and 'OK' buttons.



New Window [4-3]

This command creates a new window on the screen that initially contains no cell.

Delete Window [4-3]

This command deletes the current window from the display.

Window Partitions [4-3]



These commands control the partitioning of a window into multiple cell displays.

Split

This command causes the current editing window to split in half, with the currently edited cell appearing in both halves. Each half may be split again and again, producing an arbitrary number of nonoverlapping windows. When a window is first split, the direction of split is determined by the aspect ratio of the contents. Once split, however, the direction alternated between horizontal and vertical. When the editing window is divided, only one partition is the "current" window, as shown with a green outline. Be careful about the commands that you issue to be sure that the correct window is affected. Generally, the current window switches to whichever one has the cursor.

Delete

This command causes the current partition of the editing window to be joined with its neighbor, thus deleting the subwindow.

Make 1 Window

This command causes all partitions of the editing window to be deleted, returning to a single window with the currently edited cell.

Adjust Position [4-3]



These commands control the layout of windows on the display.

Tile Horizontally



Tile Vertically

This command causes the editing windows to be arranged horizontally, one above the other.

This command causes the editing windows to be arranged vertically, one next to the other.

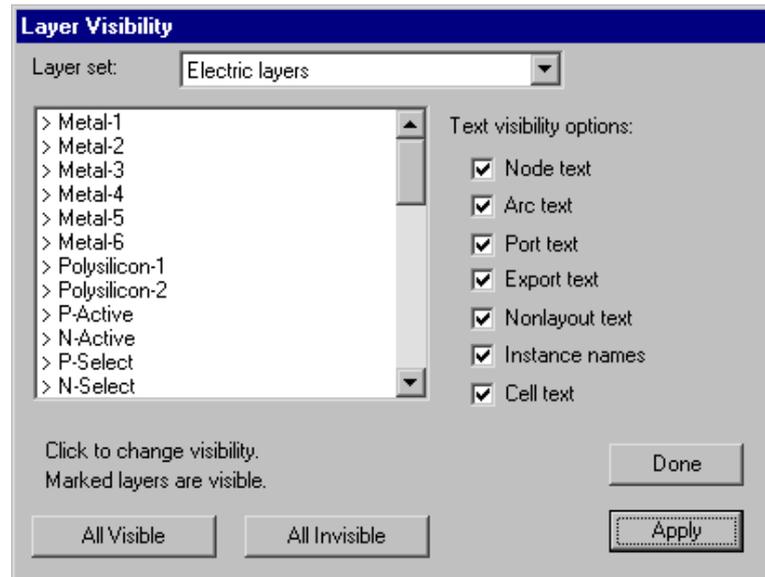
Cascade

This command causes the editing windows to be cascaded, one overlapping the other.

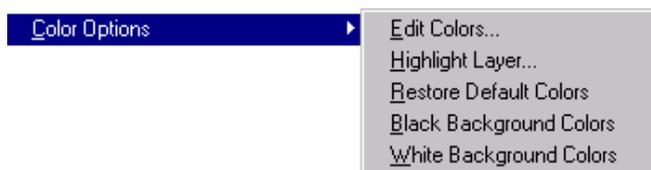
Layer Visibility... [4-5]

This command presents a dialog in which you can select the layers that are to be visible in the window. Special buttons allow all layers to be made visible or invisible. It is also possible to set visibility according to layers found in external file formats such as GDS II and DXF. Note that this dialog is modeless: it can remain up while other editing is done.

The right side of the dialog lets you choose which of the different types of text will be visible (Note that this side is titled "Text visibility options" which means that these settings are saved, whereas those on the left side are not.)



Color Options [4-6]



These commands allow you to edit the color map for the current technology.



Edit Colors...

This command displays a color wheel and a set of options for modifying the various entries.

Highlight Layer...

This command prompts for a single layer and rebuilds the color map so that the layer is highlighted. The colors can be restored with "None" button of the dialog or with the **Restore Default Colors** command below.

Restore Default Colors

This command returns the color table to its original values as specified by the current technology.

Black Background Colors

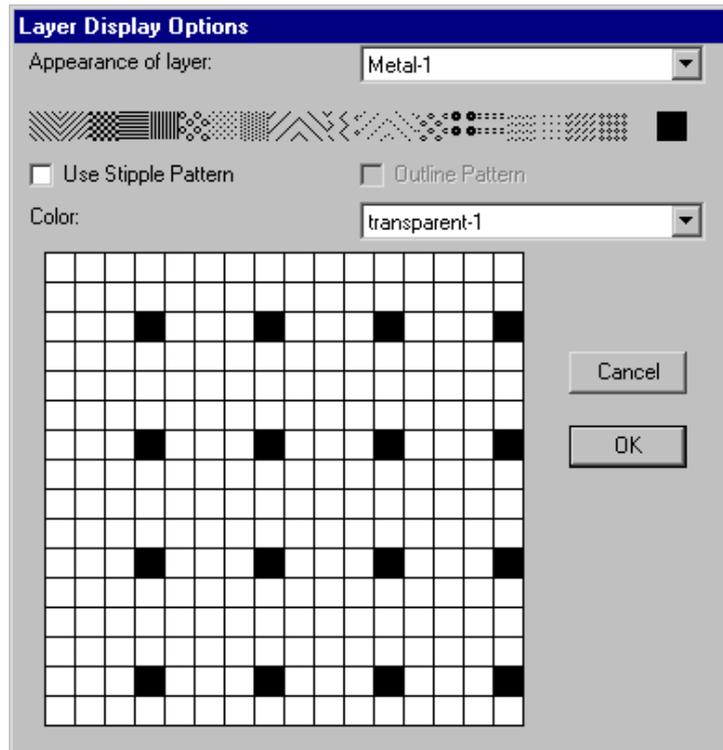
This command sets the color table to its original values as specified by the current technology, but with a black background.

White Background Colors

This command sets the color table to its original values as specified by the current technology, but with a white background.

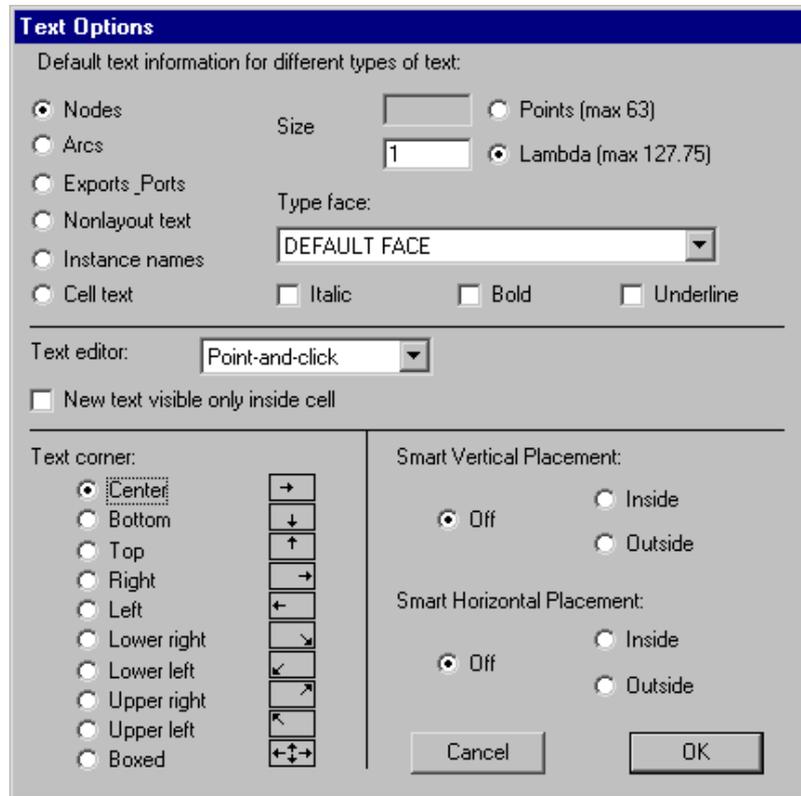
Layer Display Options... [4-6]

This command displays a dialog for examining and modifying the display of each layer. You can set individual bits in the pattern or you can choose from a set of predefined patterns along the top. You can also choose the color to use. Many layers use their stipple patterns only for printing, but a check box allows you to request that the pattern be used on the display as well. Another check box requests that the stippled polygons be outlined with a solid line.

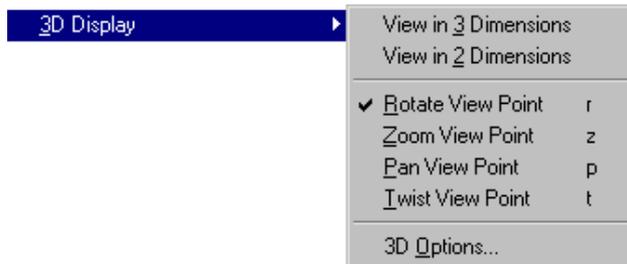


Text Options... [4–10], [6–8]

This command provides a dialog for setting the default size and placement of all subsequently created text. Text size can be absolute (in points) or relative (in lambda). The "Smart" placement options apply to export names only. The "New text visible only inside cell" option applies to text objects (created with the **Text (nonlayout)** subcommand of the **New Special Object** command of the **Edit** menu). You can also choose which text editor to use when editing large pieces of text.



3D Display [4–11]



These commands allow you to view the circuit in 3–dimensions. When viewing in 3D, you cannot edit the circuit: you can only control of the 3D view.

View in 3 Dimensions

This switches the display to a 3–dimensional view of the circuit.

View in 2 Dimensions

This switches the display back to a standard 2–dimensional view of the circuit.

Rotate View Point

This command sets rotate–mode for the 3D display, in which cursor movement rotates the objects.

Zoom View Point

This command sets zoom–mode for the 3D display, in which cursor



Pan View Point

movement up and down zooms the display in and out.

This command sets pan-mode for the 3D display, in which cursor movement shifts the display.

Twist View Point

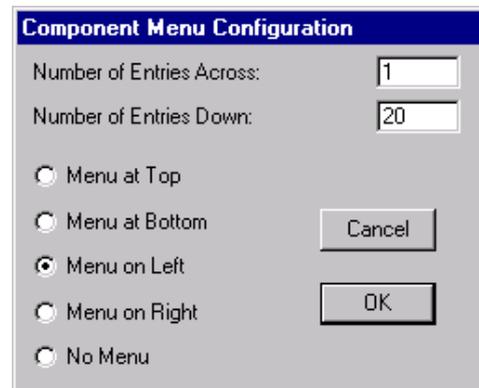
This command sets twist-mode for the 3D display, in which cursor movement rotates the display of the objects.

3D Options...

This command provides options for the 3D display, including the depth and thickness of individual layers, and whether or not to use perspective.

Component Menu... [4-8]

This command provides a dialog for adjusting the location and size of the components menu, which is initially on the left. Note that the number of entries is chosen for each technology to include the necessary nodes and arcs. If the new size contains too few entries, some components will be unavailable. If the new size contains too many entries, extra menu items will appear that may be undefined in function.



Messages Window [4-2]



These commands control the messages window.

Set Font...

This command provides a dialog for selecting the font and size of the text in the messages window.

Save Messages

This command causes all subsequent text displayed in the messages window to be saved to disk in the file "emessages.txt".

Clear

This command removes all text from the messages window.

Save Window Location

This command causes the location of the Messages window to be saved with the options (this is not needed on the Macintosh because changes to the



Messages window are always remembered).

 Previous

 Table of Contents

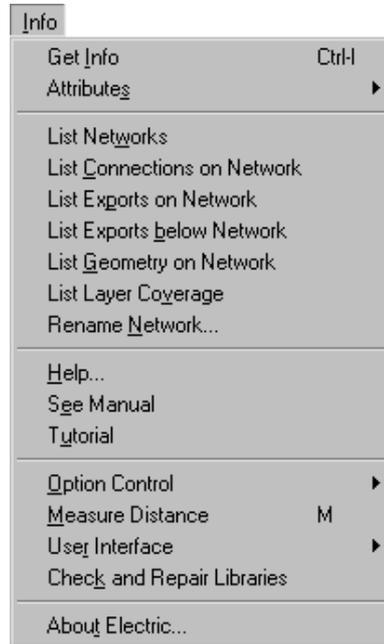
Next 



Chapter 12: MENU SUMMARY



12-8: The Info Menu



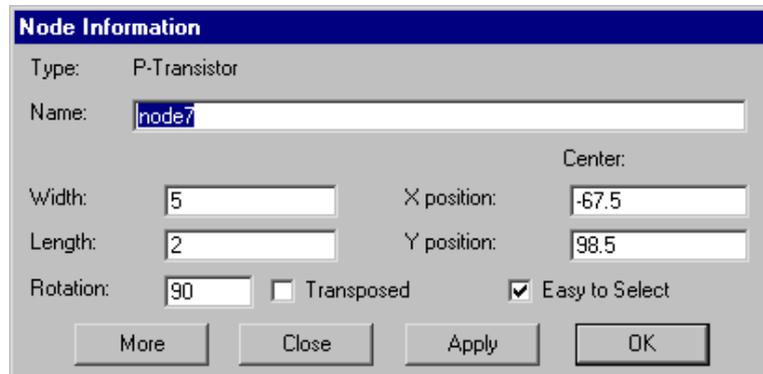
This menu provides a collection of information gathering facilities. The most useful is **Get Info** which describes the currently highlighted object. Other informational commands exist to describe the circuit. **Help...** and **Tutorial** functions are available. Also, default node and arc creation information can be set.



Get Info [2-4] [6-10] [2-5] [5-3] [3-6] [6-8]

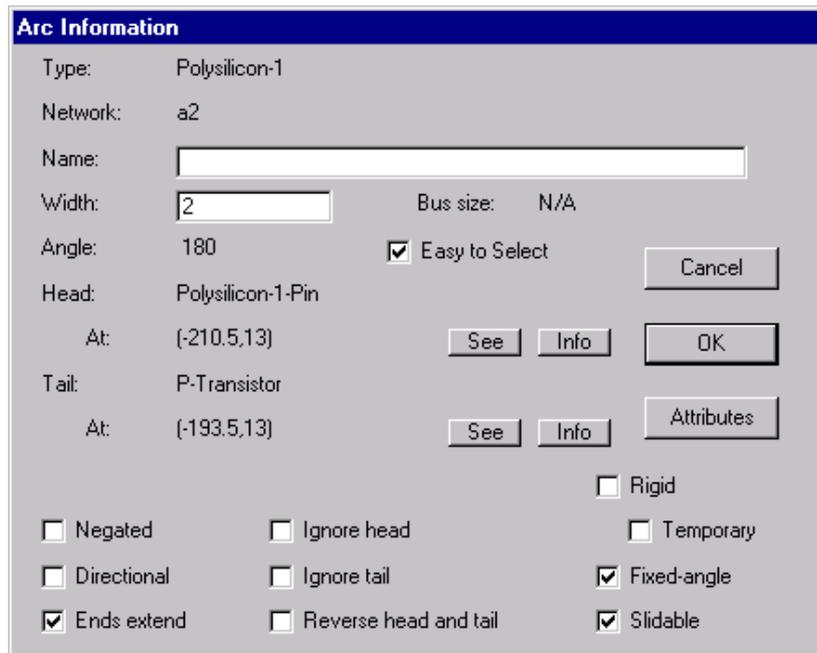
This command presents a dialog that shows information and allows modification of the currently highlighted node, arc, export, or text.

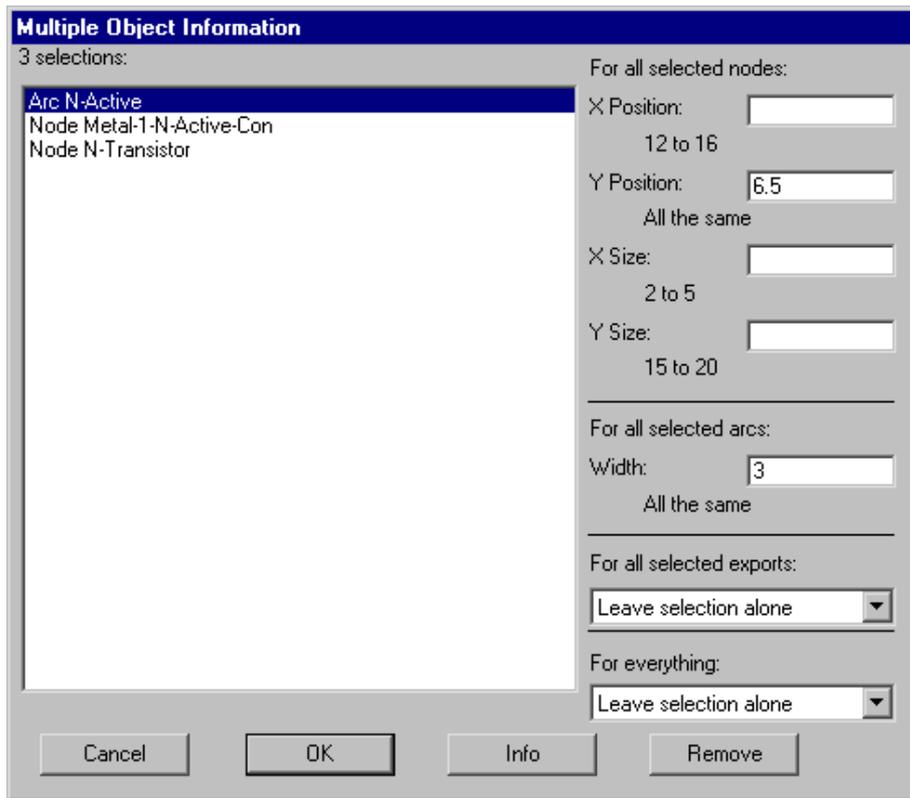
When a single node is highlighted, this command presents the dialog shown here. This dialog is modeless and can remain on the screen while other editing is done. It always shows information about the currently selected node. The dialog can grow to include more information if the "More" button is clicked.



Primitive nodes can have their size altered. The "Name" field is simply a documenting message that is displayed on the node. If you uncheck the "Easy to Select" button, you will have to use the *special select* button to select it in the future.

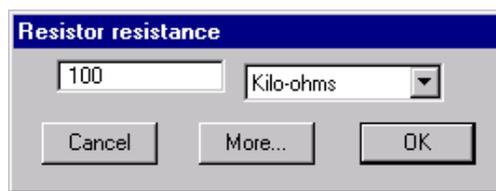
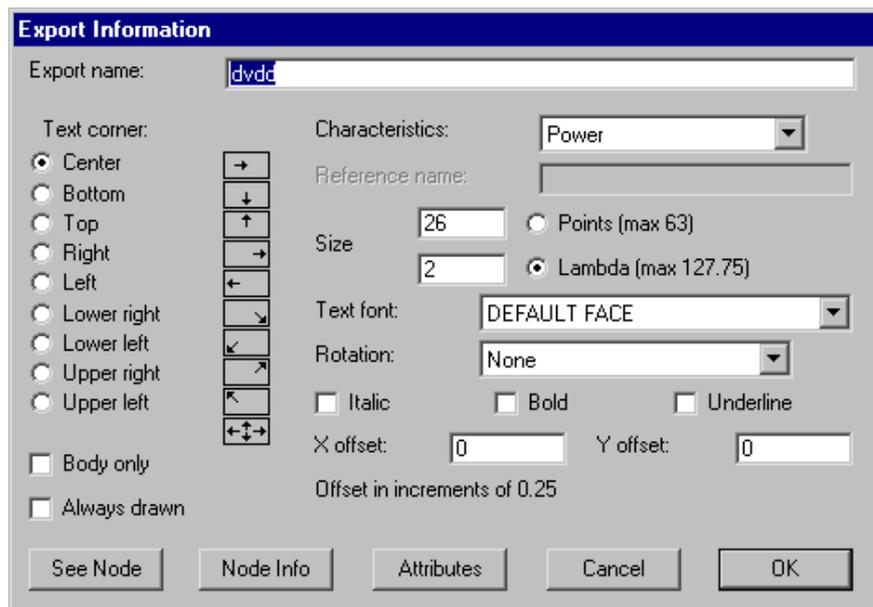
When a single arc is highlighted, this command presents the dialog shown here. All of the options in the **Arc** menu may be set here. The "Name" field defines a network name for this arc and all others electrically connected to it. You can highlight the nodes on either end of the arc with the "See" buttons, and get information about them with the "Info" buttons. If you uncheck the "Easy to Select" button, you will have to use the *special select* button to select it in the future. The "Attributes" button lets you edit special attributes on the arc.





When multiple objects are selected, the dialog lists them and lets you do simple operations on all of them (position, size, width). If two objects are highlighted with *selection* and *toggle select* buttons, this dialog also shows the distance between their centers. A popup entry allows you to make the selected objects easy or hard to select (see [Section 2-1](#)).

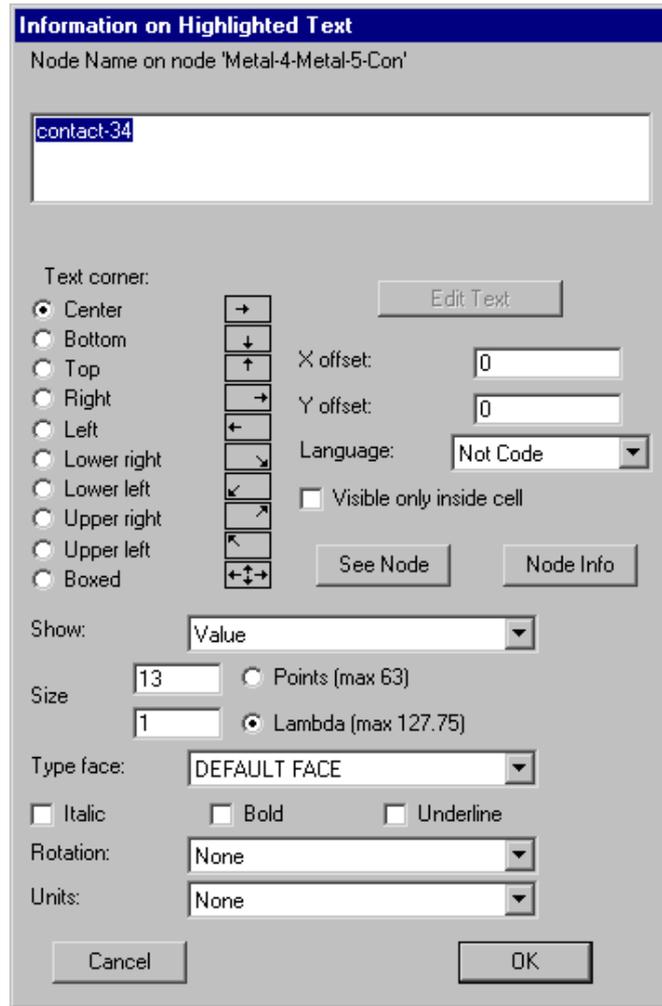
When an export's name is highlighted, this command presents the dialog shown here. The characteristics of the export can be set; the size, font, style, and rotation of the export name can be set; and the location of the export name relative to the export center can be set. The "Attributes" button lets you edit special attributes on the export.



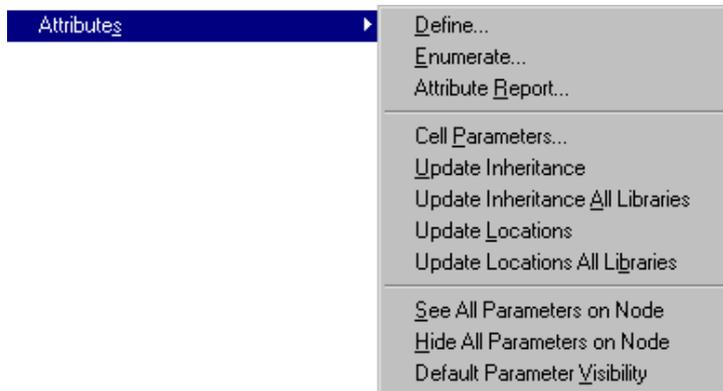
When special pieces of text are highlighted, this command presents a customized dialog for that value. For example, when the resistance value on a resistor is selected, this dialog is used.



When any other text is highlighted, this command presents the dialog shown here. Many factors can be set, including the size, the location relative to the grab-point, and whether to show the name of the attribute that holds this text.



Attributes [6–8]

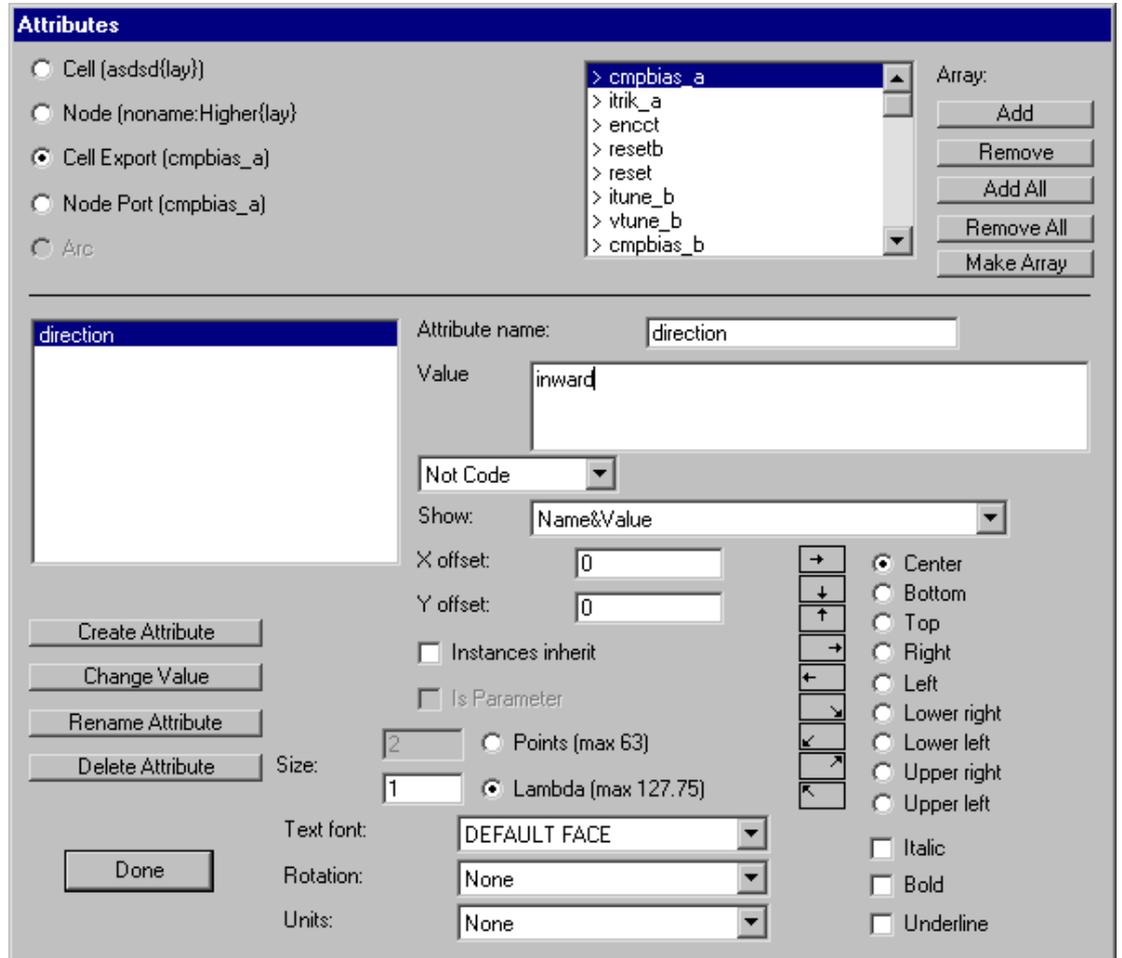


The subcommands of this command control additional attributes which can be placed on objects.



Define...

This command allows additional attributes to be placed onto parts of the circuit. The five possible locations for attributes are selectable in the upper-left. You can place attributes on (1) the current cell, (2) the currently selected node, (3) exports on the current cell, (4) ports on the currently selected node, and (5) the currently selected arc. When you choose the object on which attributes are being set, you will see a list of attributes in the lower-left. If you have chosen port or export attributes, a list of port/export names is shown in the upper-right.

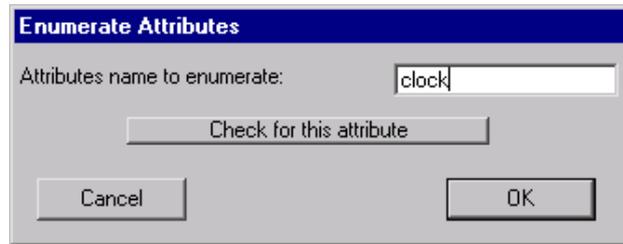


To create a new attribute, enter its name and value, and click "Create Attribute". To modify an attribute, select it and make the appropriate change. The "New Value" button changes the value, and the "Rename Attribute" button changes the attribute name. Other changes include the size (in absolute or relative units), what is shown (whether the attribute name and/or value is displayed), The offset of the text from the object to which it attaches, the grab point of the text (how it attaches to the object), and whether it is code that must be interpreted. For cell and export attributes, you can check "Instances Inherit" to make instances of these cells automatically inherit the attributes upon creation. Inheritable attribute values can use "++" or "--" to automatically increment or decrement the inherited value (for example, if an inherited attribute has the value "U12++" then the instance will inherit the value "U12" and the prototype attribute will be modified to be "U13++"). The "Delete Attribute" button removes an attribute. The "More" button presents a highly-advanced dialog for exploring the internals of the Electric database (this dialog is not documented, and the button should not be used unless you know what you are doing). The buttons in the upper-right control array of

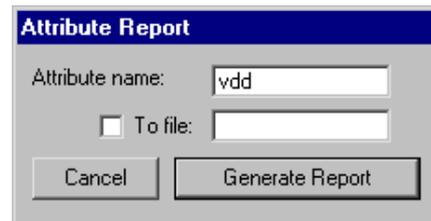


attributes from one port/export to others. By selecting names and clicking "Add" or "Remove", those port/exports are checked or unchecked in the upper-right list (the "Add All" and "Remove All" control all names). Finally, when a set of ports or exports has been selected, you select an attribute and click "Make Array" to replicate that attribute to all that are selected.

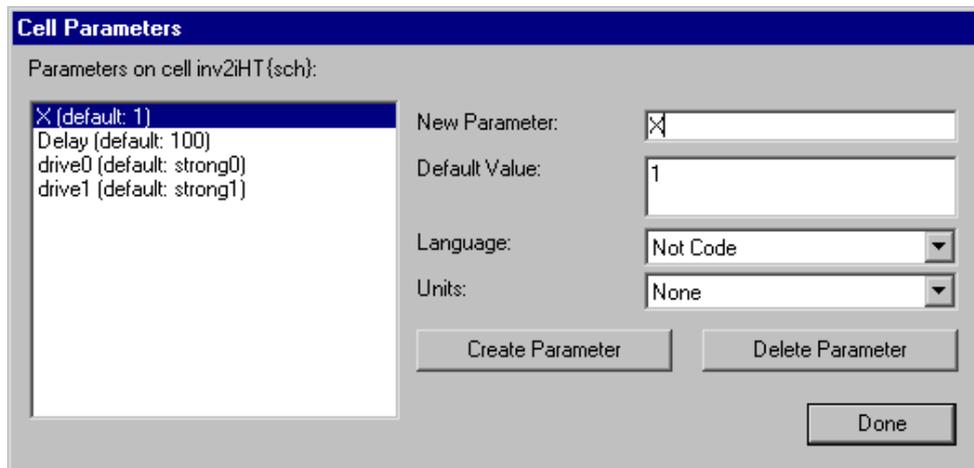
Enumerate... This command finds all occurrences of a particular attribute name and updates the attribute values. Where there is a "?" in the attribute value, it is replaced by a number that is unique.



Attribute Report... This command searches for all attributes of a given name. You can request that the list be saved in a file.



Cell Parameters This command creates parameters in the current cell. Parameters are attributes inside the cell that get copied to all instances of the cell.



Update Inheritance This requests that the selected nodes inherit all inheritable attributes. Those attributes that have already been inherited will not be altered; only new inheritable attributes will be brought from the prototypes to the instances.

Update Inheritance All Libraries This requests that all nodes inherit all inheritable attributes. Those attributes that have already been inherited will not be altered; only new inheritable attributes will be brought from the prototypes to the instances.

Update Locations This requests that the selected nodes adjust the location of all inheritable attributes to match the attribute locations on their example icon (in the schematic).

Update Locations This requests that all nodes adjust the location of inheritable attributes to match the attribute locations on their example icon (in the schematic).



All Libraries

See All

Parameters on Node This command sets all parameters on the selected nodes to be visible.

Hide All

Parameters on Node This command sets all parameters on the selected nodes to be invisible.

Default

Parameter Visibility This command sets all parameters on the selected nodes to be visible or invisible, according to their original definition in the prototype cell.

List Networks [6–9]

This command lists the named networks in the current cell. Networks can be given names by selecting an arc on the network, using the **Get Info** command above, and filling in the "Name" field.

List Connections on Network [6–9]

This command lists the active components connected to the current network. The current network is the collection of nodes and arcs connected to the currently highlighted objects. If a node is highlighted that has multiple networks on it, the particular port that is highlighted on the node determines the network.

List Exports on Network [3–7] [6–9]

This command lists all exports on the current network, at all levels of the circuit hierarchy, above and below the current cell. It is useful when checking to see that an export is named uniformly throughout the chip, because all export names are shown.

List Exports below Network [3–7] [6–9]

This command lists the exports on the current network and in all cells below this in the hierarchy.

List Geometry on Network [6–9]

This command computes the geometry connected to the current network and reports the area and perimeter of all connected layers. Only geometry at the current and lower levels of hierarchy are considered.

List Layer Coverage [3–7]

This command computes, for each layer, the percentage of the cell that is covered by that layer. This is useful because certain fabrication lines insist on a minimum amount of area for each layer.



Rename Network... [6-9]

This command presents a list of networks in the current cell and lets you rename them.

Help... [1-10]

This command provides information about the use of Electric. A dialog helps to select the subject.

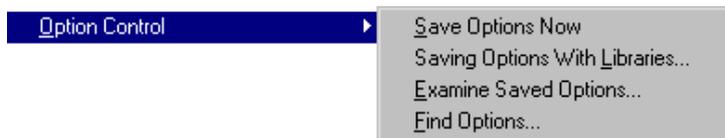
See Manual

This command displays the user's manual in a browser.

Tutorial [1-10]

This command loads a tutorial package that provides some examples of use.

Option Control [6-3]

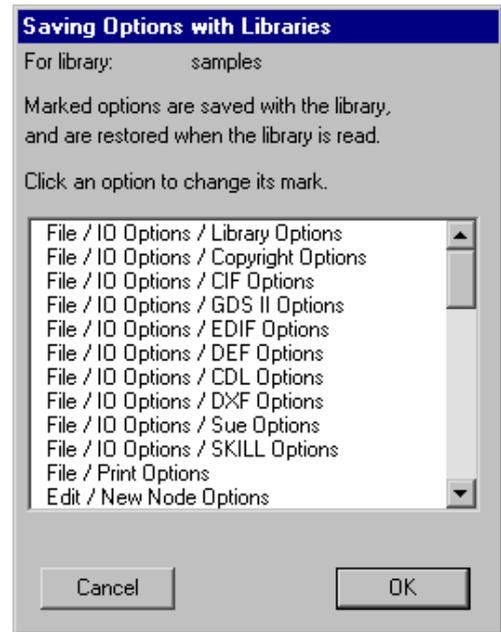


The subcommands of this command control how options, set by the many **Option** commands, are saved.

- | | |
|---|---|
| Save Options Now | This command requests that the current set of options be saved now, rather than waiting for the program to exit. |
| Saving Options With Libraries... | This command requests that certain Options be saved with the current library. An example of the usefulness of this is when a library of standard cells, designed for the Silicon Compiler, wants to store Silicon Compiler options in it so |



that the user of the library can have the proper options set.



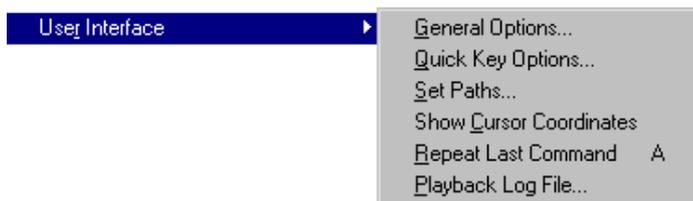
Examine Saved Options...
Find Options...

This command shows all options that are part of the saved options, indicating those that have been changed this session. This command helps you find an option. It presents a list of all options and lets you search for a keyword.

Measure Distance [4-7]

This command allows you to measure the distance between any two points on the display. After issuing it, click in the circuit to set the "starting point". Then click repeatedly in the circuit to define the "ending point" and see the measured distance. To end distance measurement, use this command again. The measured distance can be used by the **Array...** command to specify spacing. Note that this command is bound to the "M" key (shift-M).

User Interface



The subcommands of this command control various aspects of the user interface.

General Options... [9-1]

This command presents a dialog with options that pertain to the overall running of the system.

Quick Key Options... [1-9]

This allows you to rebind the quick keys (the keys that invoke menu entries).

Set Paths...

This lets you examine and modify the current directory in which library files can be found.



Show Cursor Coordinates [4–7]

This causes the cursor coordinates to be continuously displayed in the status area (they are shown where the Technology and Lambda information used to be). Issue the command again to remove coordinate display.

Repeat Last Command [1–9]

This causes the last command that was issued to be issued again. It only applies where sensible. If the last command required a dialog, that dialog is brought back, but the values entered into it must be reentered.

Playback Log File... [6–12]

This replays a session log file, which is useful for recreating lost circuitry after a crash. Electric can usually detect when a crash has occurred, so you should not normally need to issue this command.

Check and Repair Libraries [6–12]

This command examines the database for inconsistencies and repairs them whenever possible. Given that Electric is a stable, working program, this command should not uncover any problems. If, however, the system is acting strangely, try saving your library and running this.

About Electric...

This command displays a dialog with information about the current version of Electric.

 [Previous](#)

 [Table of Contents](#)

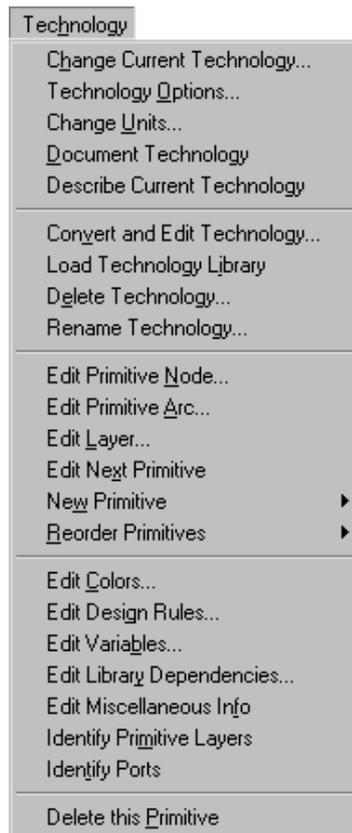
[Next](#) 



Chapter 12: MENU SUMMARY



12-9: The Technology Menu



Technologies are collections of primitive nodes and arcs built out of layers of geometry. Layers have color, design-rules and I/O correspondences. Nodes and arcs have geometry and other information.

A large set of technologies comes with Electric. These include many varieties of MOS as well as Schematics, Artwork, and more. A single cell may contain components from different technologies, simply by switching technologies and placing new components alongside the old ones. A Generic technology exists for making cross-technology interconnect as well as other special-purpose functions.

Besides being able to switch technologies and set information about the current technology, there is a technology editor that can make major changes and create new technologies. This editor works by converting a technology into a library of cells and back again. Standard editing techniques are then used on the library.



Change Current Technology... [7-1]

This command allows you to switch the set of primitives that are used in editing. A list of technologies will be presented. After selection, the menu on the left will change to show the primitives in the new technology. Note that the current technology is shown in the status area under the heading "TECHNOLOGY".

Technology Options... [7-1]

This command displays a dialog for technology-specific options. The MOSIS CMOS, Artwork, and Schematic technologies can be affected.

The screenshot shows the "Technology Options" dialog box, which is divided into two main sections: "MOSIS CMOS:" and "MOSIS CMOS Submicron (old):".

MOSIS CMOS:

- Metal layers: 4 Layers (dropdown)
- Radio buttons: SCMOS rules (4 metal or less), **Submicron rules**, Deep rules (5 metal or more)
- Checkboxes: Second Polysilicon layer, Disallow stacked vias, Alternate Active and Poly contact rules, Show Special transistors
- Radio buttons: **Full Geometry**, Stick Figures

MOSIS CMOS Submicron (old):

- Metal layers: 4 Layers (dropdown)
- Checked checkbox: Automatically convert to new MOSIS CMOS

Artwork:

- Unchecked checkbox: Arrows filled

Schematics:

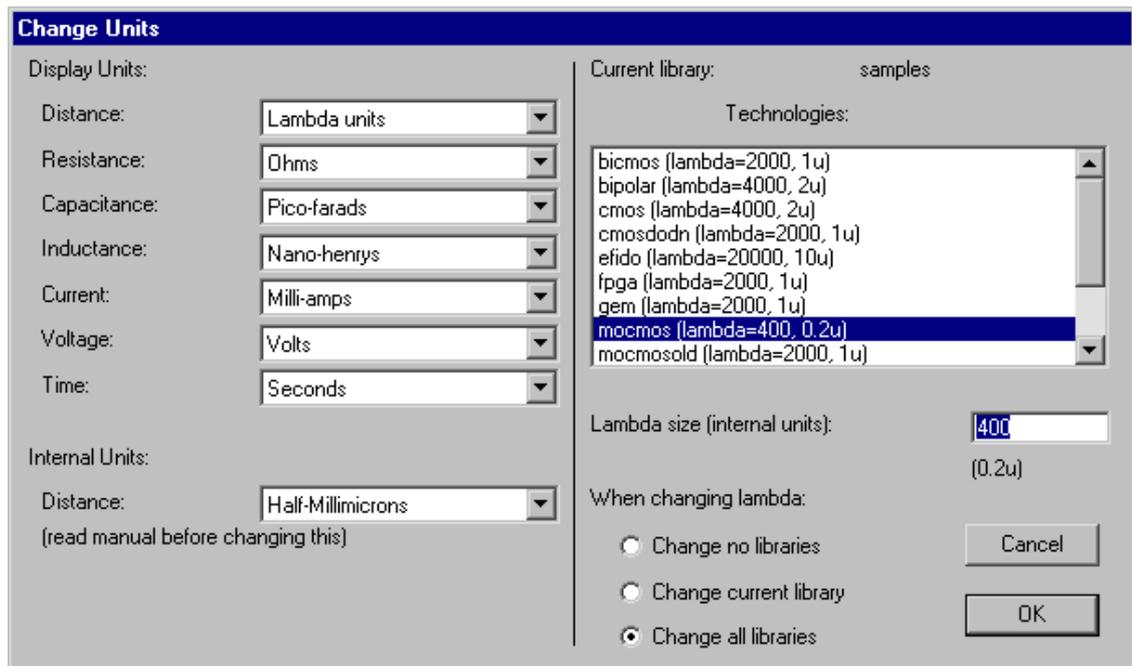
- Negating Bubble Size: 1.2 (text input)
- Use Lambda values from this Technology: mocmos (dropdown)

Buttons: Cancel, OK



Change Units... [7-2]

The left side of the dialog allows you to change the displayed units for various electrical properties. You can also change the internal unit size (be very careful when changing this).



The right side of this dialog allows you to change the value of "lambda" for any technology. Lambda is the number of internal units per grid unit (the current value, in microns, is shown in the status area under the heading "LAMBDA"). When changing this value, you can choose to scale all objects (which will keep them looking the same as before, only scaled), or to just update the technology (which will alter the appearance of existing objects). You can also choose to make this change to all libraries or just the current one. Do not use this command to adjust the lambda size when editing technologies: use the **Edit Miscellaneous Info** command instead.

Document Technology [7-1]

This command writes a disk file with a textual description of a technology. The description lists layers, nodes, and arcs, and other relevant information. The file is for descriptive purposes only and cannot be read back into Electric.

Describe Current Technology [7-1]

This command displays information about the current environment of design.

Convert and Edit Technology... [8-2]

This command allows an existing technology to be modified using the technology editing facilities. You will be prompted for the technology to edit. This technology will then be converted to a library of the same name, and that library will become the current library. Editing any cell of this library will be done in *technology*



editing mode, which means that a new button is defined for making changes to the highlighted object: *technology edit*. The remainder of the commands in the **Technology** menu apply to technology editing.

Once in technology editing mode, this menu entry changes to **Convert Library to Technology...** so that the library can be translated back into a new technology and the mode can be terminated. This command presents a dialog for creating a new technology from the library. Note that the new technology name may have to be changed because there will probably already be one with that name. After creating the technology, you may switch to it with the **Change Current Technology...** command above. To save an edited technology, use the **Save Library** command of the **File** menu, just as you would save any other library. The library can then be restored with the **Load Technology Library** command below.

Load Technology Library [8-2]

This command causes an existing technology library to be read from disk and converted to a technology.

Delete Technology... [8-2]

This command prompts for a technology name and deletes it. There cannot be any uses of the primitives in that technology or else the command will fail.

Rename Technology... [8-2]

This command allows you to rename the current technology.

Edit Primitive Node... [8-8]

This command causes a primitive node from the currently edited technology to be brought into the window. Because each primitive node is represented as a cell of the library, it is also possible to edit a node by using the **Edit Cell...** command in the **Cells** menu. However, the node names are somewhat encoded in the library, so it is more convenient to use this command.

The cell describing a primitive node contains four examples of the node, scaled in both axes. These images are built from separate layers which may be selected, edited, etc. (use the *technology edit* button to change what is highlighted). In addition to the geometry layers, there is also a highlight layer which shows where the highlighting will appear when the primitive is selected. Finally, there are port layers that show where arcs will connect to the primitive. In the upper-left (least scaled) version of the node, the ports have additional information such as arc connectivity and connection angle ranges.

Besides the four examples, there are special messages on the right that set special attributes of the node.

Edit Primitive Arc... [8-7]

This command causes an arc from the currently edited technology to be brought into the window. Because each arc is represented as a cell of the library, it is also possible to edit an arc by using the **Edit Cell...** command in the **Cells** menu. However, the arc names are somewhat encoded in the library, so it is more convenient to use this command.



The cell describing an arc contains one example of the arc, laid out horizontally. The length of this arc is not important. On the right are special messages that describe the arc. To change an object, highlight it and use the *technology edit* button.

Edit Layer... [8-5]

This command causes a specified layer from the currently edited technology to be brought into the window. Because each layer is represented as a cell of the library, it is also possible to edit a layer by using the **Edit Cell...** command in the **Cells** menu. However, the layer names are somewhat encoded in the library, so it is more convenient to use this command.

The cell describing a layer contains a stipple pattern and a color box for describing the appearance of the layer. Special information is shown on the right, including the color to use and I/O correspondences for the layer. The stipple pattern is used only if the color is not a "Transparent" one. To change anything in the cell, highlight it and use the *technology edit* button.

Edit Next Primitive [8-5] [8-7] [8-8]

This command edits the next primitive node, arc, or layer (whichever is currently being edited). If the last primitive is being edited, the first is displayed.

New Primitive [8-5] [8-7] [8-8]



The subcommands of this command create new cells to describe a primitive node, arc, or layer in the currently edited technology.

Reorder Primitives [8-5] [8-7] [8-8]



The subcommands of this command produce lists of primitive nodes, arcs, and layers in an edit window and allows you to rearrange the order in which they will appear in the technology.

Edit Colors... [8-6]

This command causes the color map for the edited technology to be presented for modification.

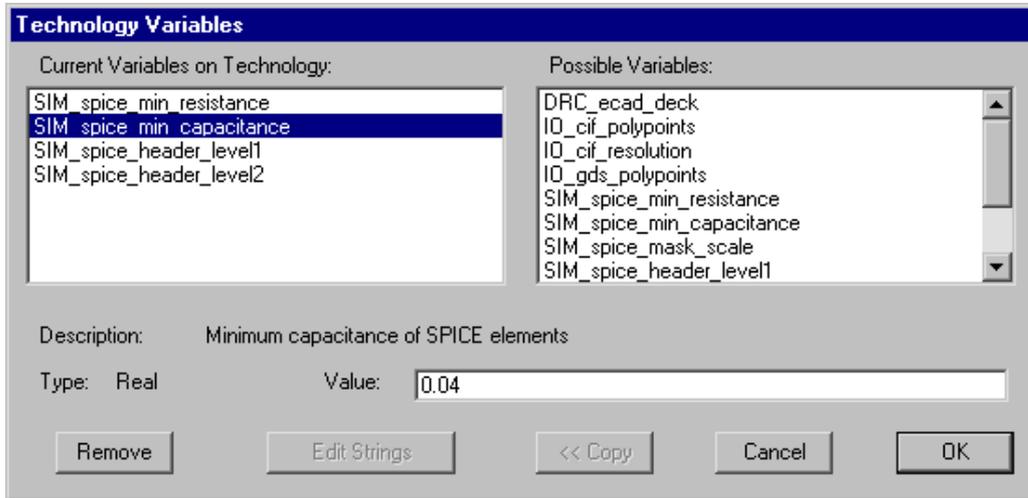
Edit Design Rules... [8-6]

This command causes the design rules for the edited technology to be displayed in a text editing window. Each design rule consists of a spacing distance and two layer names. If the spacing distance is preceded by a "c", this rule applies only when the two layers are electrically connected.



Edit Variables... [8-4]

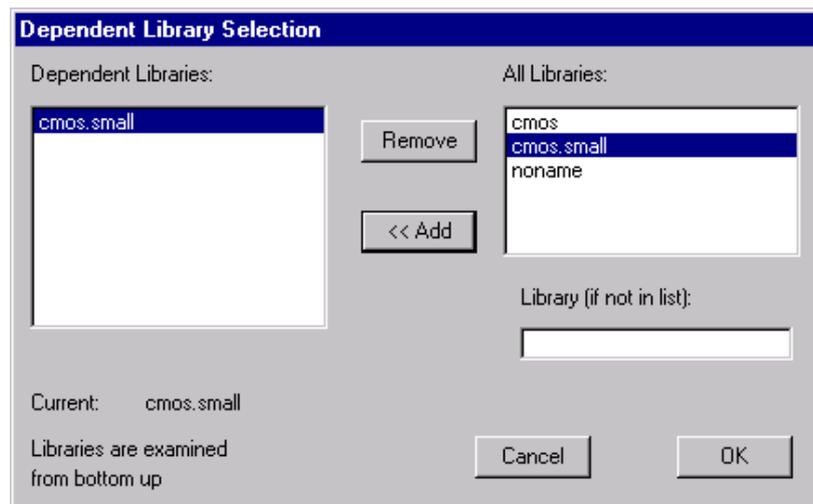
This command allows you to specify additional pieces of information about the technology being edited. A dialog shows two lists of variables that can be attached to the technology. On the left is a list of variables that are currently attached, and on the right is a list of all known variables that can be attached to the technology. To add a new variable to the current list, select it in the list of possibilities and click the "<< Copy" button. To remove a variable from the technology, select it in the left-hand list and click the "Remove" button. When a variable in either list is selected, its description and type are shown. When the variable is of type "Integer", "Real", or "String", you may change its value in the "Value" field. When you have selected a variable of type "Strings" (note the final "s") then it must be edited in a separate window with the "Edit Strings" button. The "Edit Strings" button first exits this dialog and then opens a text edit window for manipulating this variable. See [Section 4-10](#) for more on text editing.



Edit Library Dependencies... [8-3]

This command allows you to enter a list of technology libraries that will be combined with the current library when defining a technology. This is a list of *dependent libraries* that combine to define the technology.

The dialog contains 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.

Edit Miscellaneous Information [8-4]

This command causes miscellaneous factors from the currently edited technology to be brought into the window for modification. To change a factor, highlight it and use the *technology edit* button.

Identify Primitive Layers [8-7] [8-8]

This command causes all of the layers in the currently edited primitive node or arc to be labeled. The labels are temporary, and disappear when the window is redrawn in any way.

Identify Ports [8-8]

This command causes all of the ports in the currently edited primitive node to be labeled. The labels are temporary, and disappear when the window is redrawn in any way.

Delete this Primitive [8-5] [8-7] [8-8]

This command deletes the primitive node, arc, or layer in the current window.

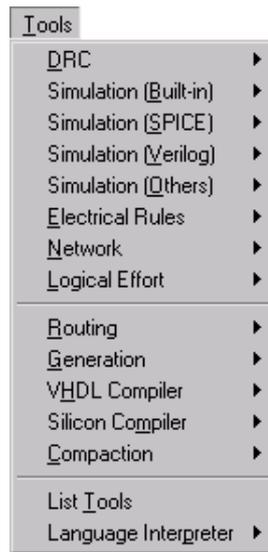
 [Previous](#)  [Table of Contents](#) [Next](#) 



Chapter 12: MENU SUMMARY

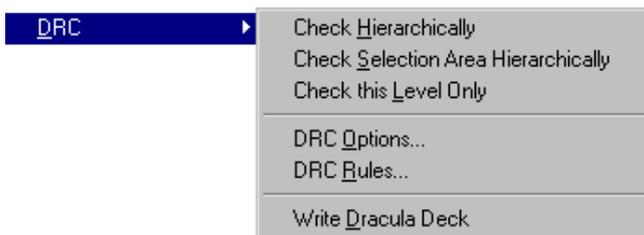


12-10: The Tools Menu



This menu is a collection of submenus that controls the different analysis and synthesis tools in Electric. For analysis, there are Design–Rule Checkers, a simulator, many simulation interfaces, and a network consistency checker. For synthesis, there are routers, PLA generators, a VHDL compiler, and a silicon compiler place–and–route system.

DRC [9-2]



This submenu controls the design–rule checkers. There is an incremental system which watches all design and displays warnings where appropriate. There are also two hierarchical checkers and an interface to the Dracula DRC system.

Check Hierarchically This checks the current cell hierarchically (all geometry is checked, all the way down the hierarchy).

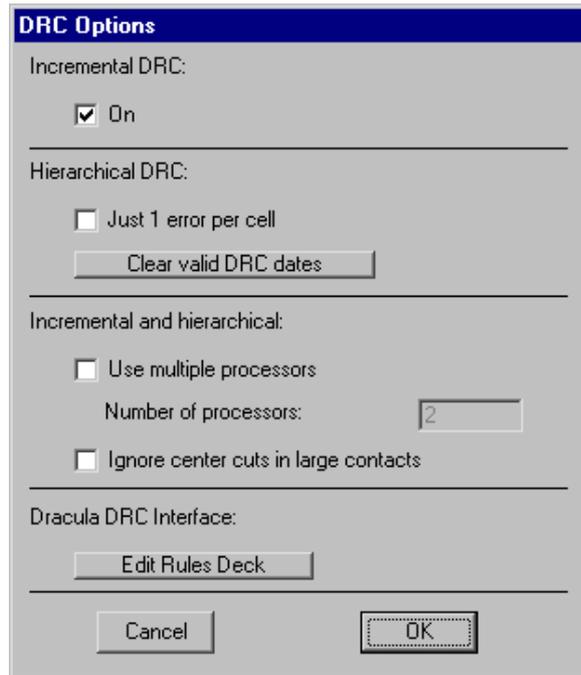
Check This checks the current cell hierarchically, but only in the selected area.



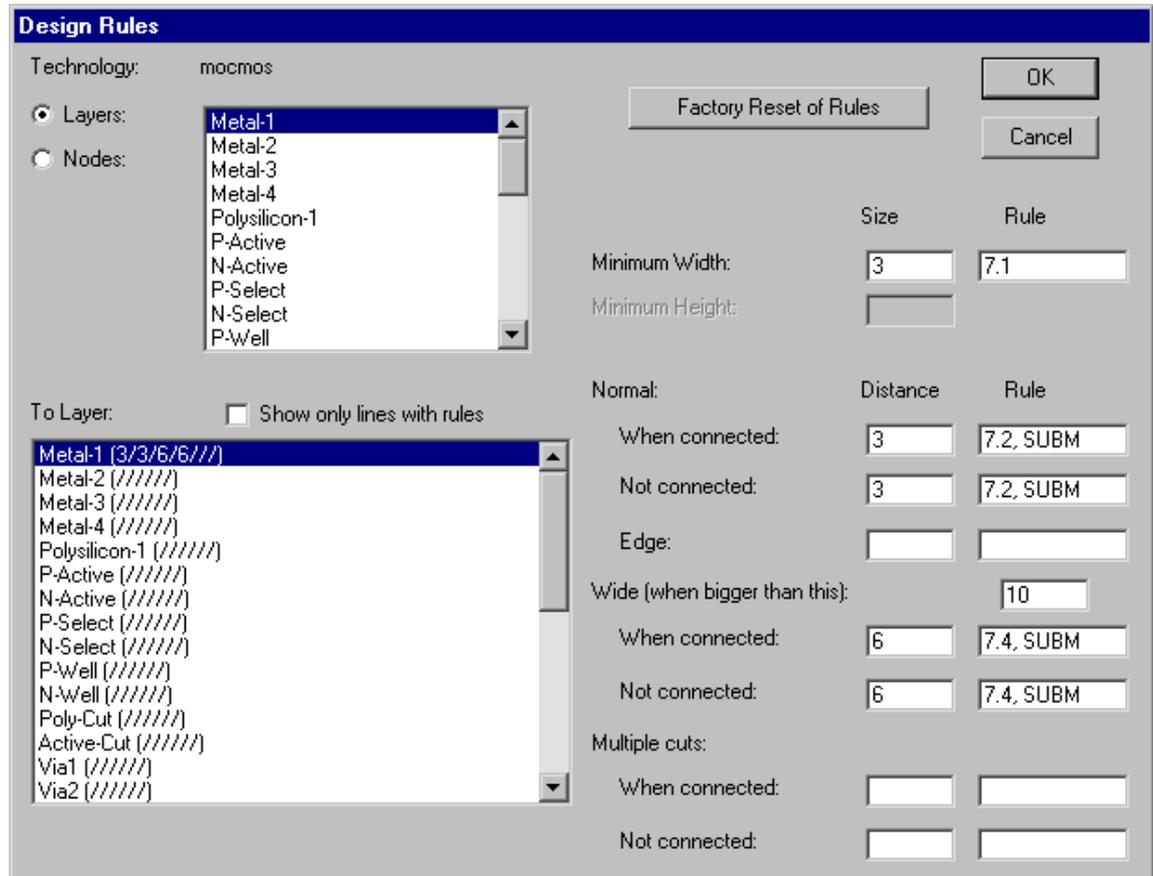
**Selection
Area
Hierarchically**

Check this Level Only This command checks the current cell nonhierarchically (only geometry in the current cell is checked, not in any subcells).

DRC Options... This command lets you control a number of DRC options, including whether or not the incremental DRC is running, options for Dracula, and more.



DRC Rules... This command provides a way to examine and modify the design rules (by using the "For layer" and "To layer" areas).

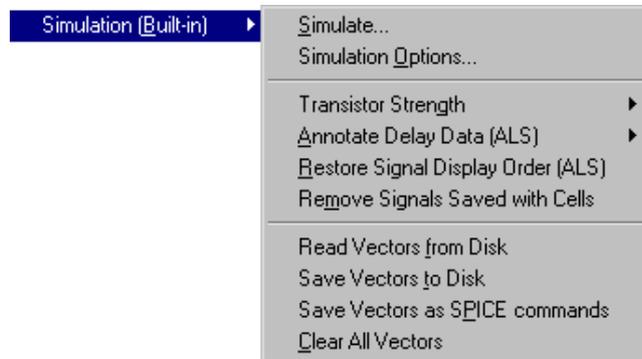


**Write
Dracula
Deck**

This command tells the design-rule checker to produce an input deck for the Dracula design-rule checker. At the current time, only layout in the Mosis CMOS (mocmos) technology can be checked in this manner. However, with the "Edit Dracula Deck" button of the **DRC Options...** dialog, rule sets may be defined for any technology.



Simulation (Built-in) [10-1] [10-2]

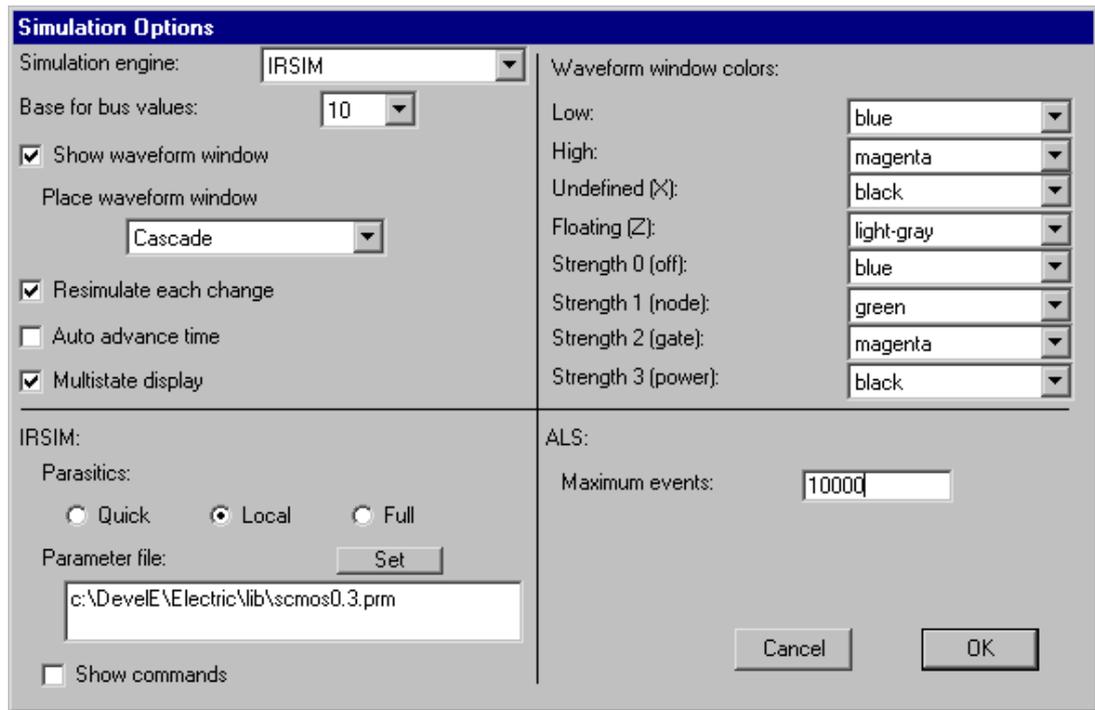


This submenu controls the gate-level simulator in Electric. Electric comes with a simulator called ALS (Asynchronous Logic Simulator), but you can also add an IRSIM engine. In addition to controlling live simulators, these commands can also be used to manipulate waveforms that come from batch simulators such as SPICE and Verilog.

Simulate... This command causes the current cell to be simulated. For ALS simulation, the cell is converted to VHDL, the VHDL is compiled into a netlist, and the netlist is simulated. For IRSIM, a netlist is generated and the simulator is invoked. A waveform display is shown for viewing signal values.

Simulation Options... This command presents a dialog for control of simulation parameters. The simulation engine can be selected (initially, Electric comes with only the ALS simulator, but a built-in IRSIM engine is available from Static Free Software). The "Resimulate each change" item causes each change that is made to something being simulated to trigger resimulation and display of the results. The "Auto advance time" item tells the simulator to move the time cursor automatically when a new signal is added to the simulation. The "Multistate display" check tells the simulator to show signals in the layout or schematics window with texturing and color to indicate strength. Without this, a simple on/off indication is drawn in the layout or schematics window. The "Show waveform window" check tells the simulator to create a separate window with waveform plots when simulation starts. The waveform window can be cascaded (a separate overlapping window) or tiled (placed in one half of the original circuit's window). The radix of bus signals in the waveform window can be selected. For ALS, the maximum number of events to simulate can be changed if you want to extend the simulator's range (and memory usage). For IRSIM, the level of parasitics and the file of parasitic information can be specified.



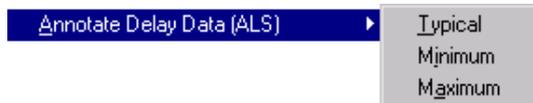


Transistor Strength



This command lets you change the strength of the selected transistor.

Annotate Delay Data (ALS)



This command lets you select which of the sets of stimulus data to use for the ALS simulator. The stimulus data is acquired with the **SDF** subcommand of the **Import** command of the **File** menu.

Restore Signal Display Order (ALS)

This command restores the default set of signals in the waveform display, which is useful if they have been rearranged or if some signals were deleted. It only works for the ALS simulator.

Remove Signals Saved with Cells

The waveform display remembers the signal names that are associated with each cell. When the user rearranges signals, it is preserved for the next time that cell is simulated. This command clears the saved list of signal names associated with every cell so that the waveform window will use a default set.

Read Vectors from Disk

This command causes a file of test vectors to be read from disk. You will be prompted for the file name.

Save Vectors to Disk

This command causes the current set of test vectors to be saved to disk. You will be prompted for the file name.



Save

Vectors as SPICE

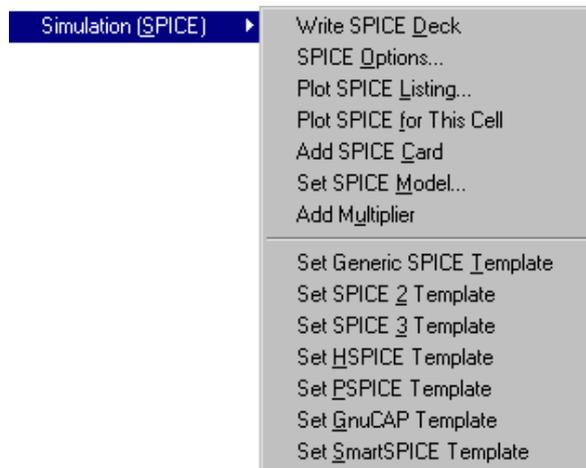
This command exports the current set of test vectors as a SPICE deck.

commands

Clear All Vectors

This command erases all test vectors from the simulation.

Simulation (SPICE) [9-4]



This submenu controls the SPICE simulator, including deck generation and plotting SPICE output.

Write SPICE Deck

This command generates an input deck for the SPICE circuit-level simulator. Because SPICE is not an interactive system, it is necessary to specify inputs and outputs in the circuit. This is done by placing Source and Meter components (from the **New Analog Component** submenu of the **Edit** menu), parametrizing them with the actual SPICE message, and connecting them to the circuitry. It is also necessary to specify Transient or DC analysis by placing an appropriate Source component in the cell.



SPICE Options...

This command allows many SPICE options to be controlled, for example, the SPICE format (SPICE 2, SPICE 3, HSPICE, PSPICE, GnuCAP, or SmartSPICE); control of parasitics in the deck; control of SPICE execution (UNIX systems only); control of header, trailer, and individual cell model cards; and much more.

SPICE Options

SPICE Engine: HSpice Use Parasitics Cancel

SPICE Level: 1 Use Node Names OK

Output format: Standard Force Global VDD/GND

Use Cell Parameters

Write Trans Sizes in Lambda

Run: Don't Run SPICE

With: _____

SPICE primitive set: spiceparts

For technology: mocmos

Layer: Metal-1 Resistance: 0.06

Metal-2 Capacitance: 0.07

Metal-3 Edge Cap: 0

Metal-4

Min. Resistance: 50 Min. Capacitance: 0.04

Use Built-in Header Cards Edit Built-in Headers for Technology/Level

Use Header Cards with extension: _____

Use Header Cards from File: _____

Browse Header Card File

No Trailer Cards

Use Trailer Cards with extension: _____

Include Trailer from File: _____

Browse Trailer Card File

For Cell: asdsd{lay} Derive Model from Circuitry

hifidely{ic} Use Model from File: Browse

hifidely{sch}

inv{ic}

Plot SPICE Listing...

This command reads the output of a SPICE run and shows the signals in a waveform window. You will be prompted for the name of the SPICE output file.

Plot SPICE for This Cell

This command reads the output of a SPICE run and shows the signals in a waveform window. This command presumes that the SPICE output file has the same name as the current cell (with appropriate extensions). For example, if you are editing cell "myclock{lay}", this command will look for file "myclock.sp".

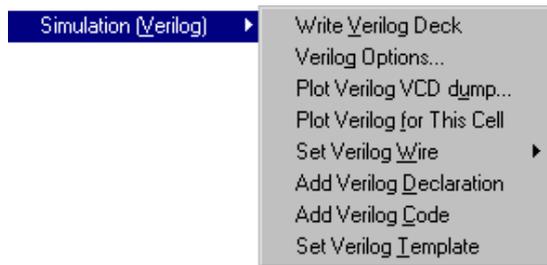
Add SPICE Card

This command allows you to click in the design and type a SPICE card that will be inserted into the generated deck.



Set SPICE Model...	This command allows you to change the SPICE model of the currently selected node.
Add Multiplier	This command places a multiplier factor on the currently selected node. This factor is used to scale the transistor sizes.
<hr/>	
Set Generic SPICE Template	This command allows you to create a SPICE template for the current cell.
Set SPICE 2 Template	This command allows you to create a template specifically for SPICE 2.
Set SPICE 3 Template	This command allows you to create a template specifically for SPICE 3.
Set HSPICE Template	This command allows you to create a template specifically for HSPICE.
Set PSPICE Template	This command allows you to create a template specifically for PSPICE.
Set GnuCAP Template	This command allows you to create a template specifically for GnuCAP.
Set SmartSPICE Template	This command allows you to create a template specifically for SmartSPICE.

Simulation (Verilog) [9-4]



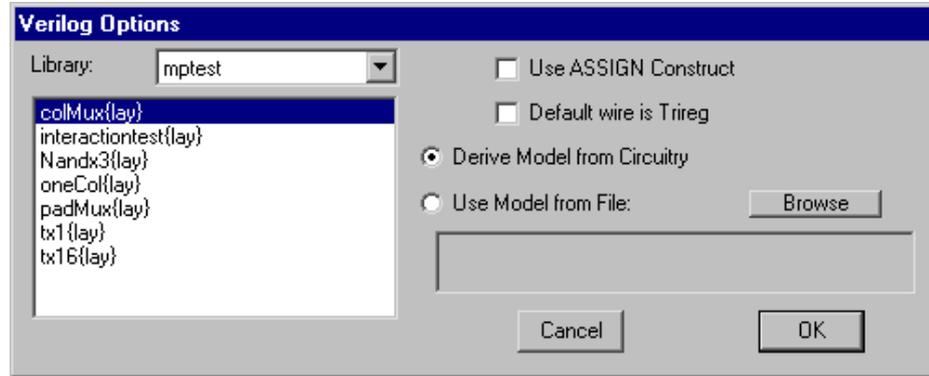
This submenu controls the generation of Verilog simulation netlists.

Write Verilog Deck This command generates an input deck for the Verilog simulator.



Verilog Options...

This command generates displays a dialog for controlling Verilog deck generation.



Plot Verilog VCD Dump

This command reads a dump file (the output of Verilog simulation) and displays the result in a waveform window.

Plot Verilog for This Cell

This command reads the output of a Verilog run and shows the signals in a waveform window. It presumes that the Verilog output file has the same name as the current cell (with appropriate extensions). For example, if you are editing cell "myclock{lay}", this command will look for file "myclock.dump".

Set Verilog Wire

This command lets you set the type of Verilog wire that the current arc will produce (either **Wire**, **Trireg**, or **Default**). The **Default** option uses the setting from the **Verilog Options...** dialog.

Add Verilog Declaration

This command allows you to click in the design and type Verilog declarations that will be inserted into the generated deck.

Add Verilog Code

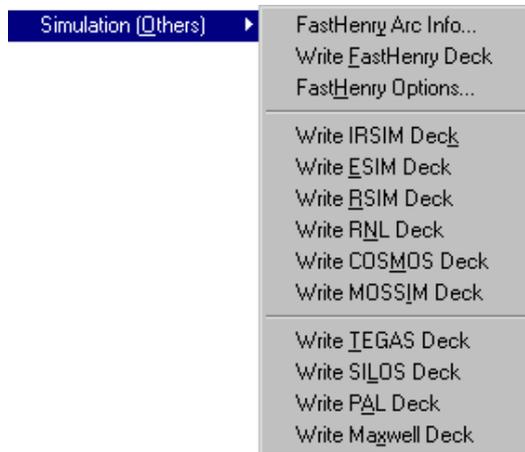
This command allows you to click in the design and type Verilog code that will be inserted into the generated deck.

Set Verilog Template

This command allows you to create a Verilog template when defining new primitives.



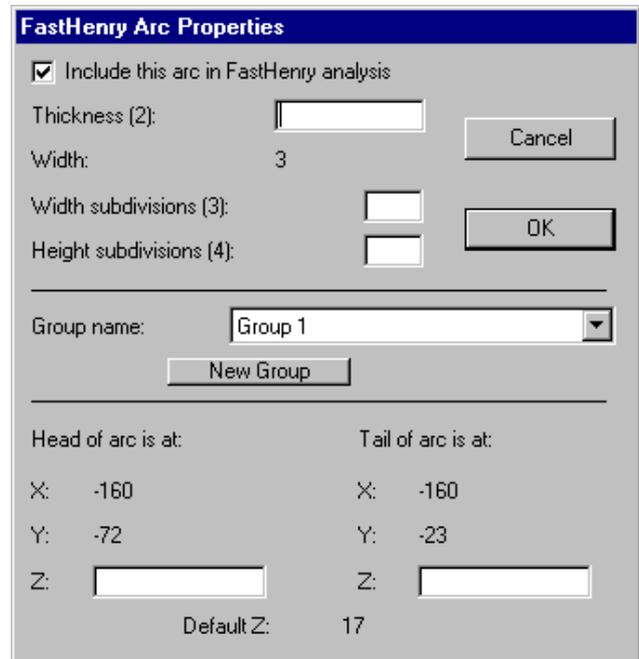
Simulation (Others) [9-4]



This submenu allows input decks to be written for many different simulators.

FastHenry Arc Info...

This command presents a dialog for including the currently selected arc in the FastHenry analysis, and for setting options on that arc. You can override the thickness, set the number of subdivisions, set the analysis group, and even set a height for the two arc ends.



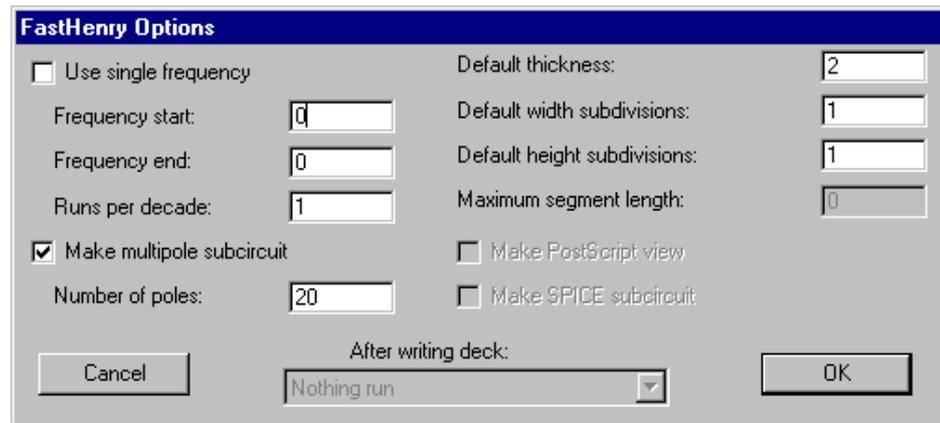
Write FastHenry Deck...

This command generates a FastHenry deck from the current cell.



FastHenry Options...

This command presents a dialog of options for FastHenry deck generation. The frequency and multipole options control flags that are placed in the deck. The default thickness and subdivision fields provide values that are used when no overrides are specified for individual arcs.



Write IRSIM Deck

This command generates an input deck for the IRSIM switch-level simulator.

Write ESIM Deck

This command generates an input deck for the ESIM switch-level simulator (nMOS only, no timing).

Write RSIM Deck

This command generates an input deck for the RSIM switch-level simulator (nMOS only).

Write RNL Deck

This command generates an input deck for the RNL switch-level simulator (nMOS only, Lisp-like interface).

Write COSMOS Deck

This command generates an input deck for the COSMOS switch-level simulator (MOS only).

Write MOSSIM Deck

This command generates an input deck for the MOSSIM switch-level simulator (MOS only).

Write TEGAS Deck

This command generates an input deck for the TEGAS/TEXSIM gate-level simulator.

Write SILOS Deck

This command generates an input deck for the SILOS simulator.

Write PAL Deck

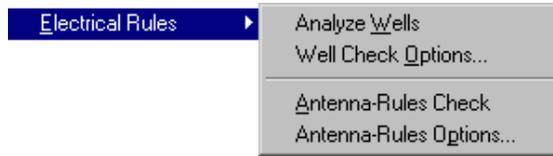
This command generates an input deck for the Abel PAL generator/simulator.

Write Maxwell Deck

This command generates an input deck for the Maxwell simulator.



Electrical Rules [9-3]



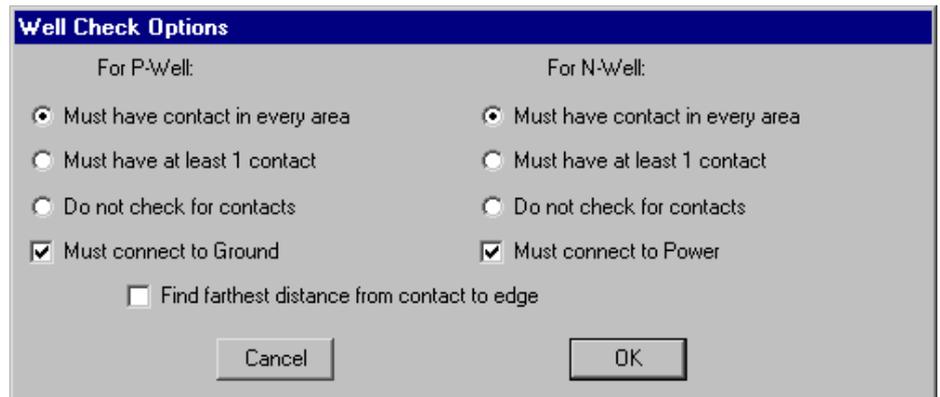
These commands do static analysis of the circuit, which include well analysis and antenna rules.

Analyze Wells

This command examines the current cell and checks all well areas for proper electrical rules. The farthest distance from a well contact to the edge of its implant is shown.

Well Check Options...

This presents a dialog with Well and Substrate Checking options. You can choose to require one contact per area, or only one contact anywhere on the chip. You can also check for proper power and ground connections.

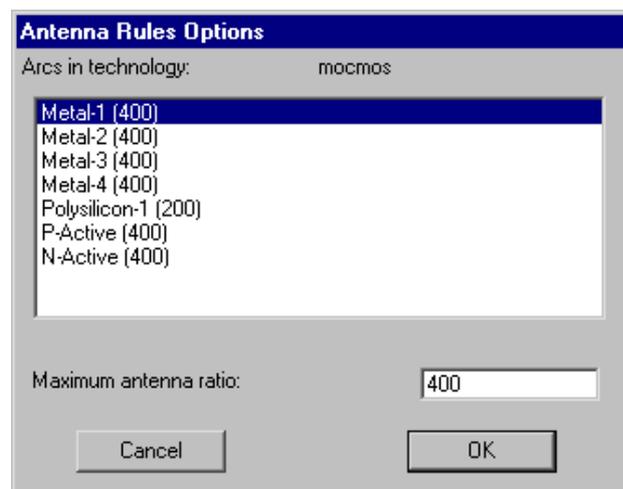


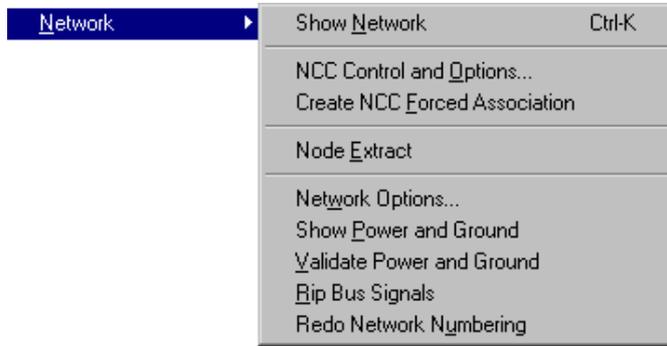
Antenna-Rules Check

This command examines the current cell and all below it for antenna rules violations. Antenna rules ensure that there is no path from a large area of metal or poly to a small number of transistor gates.

Antenna-Rules Options...

This presents a dialog with Antenna rule ratio limits.



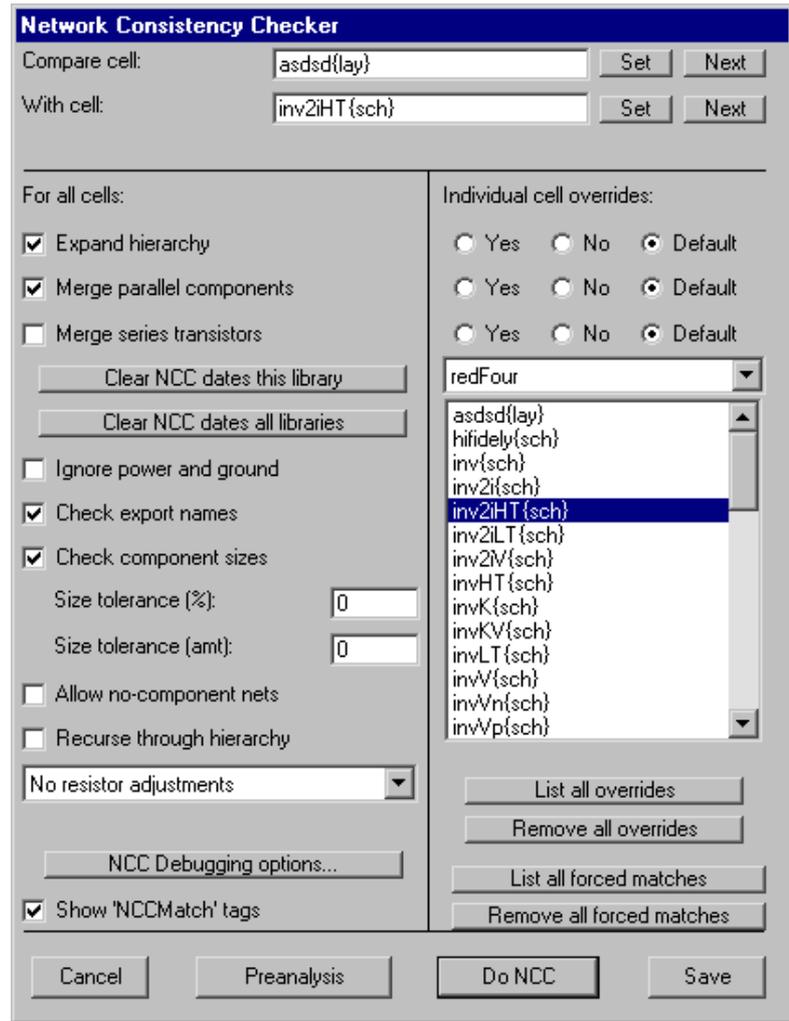


This submenu controls miscellaneous network functions, including a Network Consistency Checking (NCC) facility. Some systems call network consistency checking "LVS" (Layout vs. Schematic), but Electric uses the term NCC because it can compare any two cells, not just layout vs. schematic.

Show Network This command shows the equivalent to the currently highlighted network in all other windows. It also works for network names selected in a text window. If this cell has been run through the network consistency checker, that information will be used.

NCC Control and Options... This presents a dialog for controlling the network consistency checker. On top are the two cells whose networks are to be compared. If there are two cells currently being displayed, they are loaded into these fields.





The lower portion controls the NCC process. On the left are the NCC controls, and on the right half are per-cell overrides of some of these settings. The bottom has buttons for running NCC or Preanalysis.

Create NCC Forced Association

This command places an NCC-match tag on the currently select object. By changing the name, and setting the same name on objects in different cells, those objects are forced to match during NCC.

Node Extract [7-3]

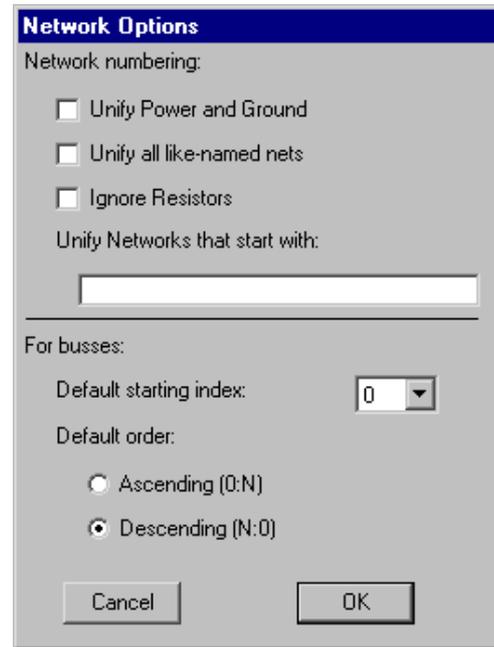
This command extracts connectivity from a cell that has only geometry (pure-layer nodes). Such cells are the result of reading external format files, such as CIF and GDS. The pure-layer nodes are removed and replaced with a connected network of nodes and arcs. Unfortunately, the node extractor is only partially implemented and should not be counted-on to properly extract. Also, it does not recognize transistors.



Network Options...

This presents a dialog for controlling the network tool. The top part has network numbering options including rules for unifying networks, and choices for handling resistors.

The lower part of the dialog controls how busses will be numbered when they are not explicitly defined. You can choose to start them at 0 or 1, and can choose to order them ascending or descending.



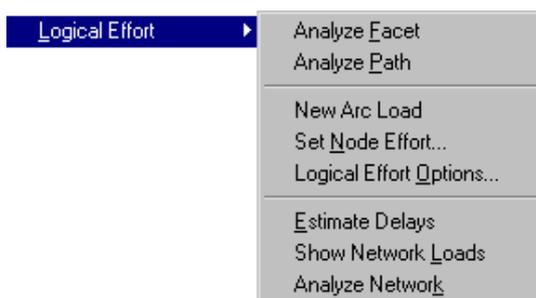
Show Power and Ground This command highlights all of the power and ground networks in the current cell.

Validate Power and Ground This command checks all power and ground networks in the circuit to be sure that they are named sensibly.

Rip Bus Signals [7-6] This command takes the currently selected bus wire and adds wire taps for each signal on the bus. The wires run perpendicular to the bus and are labeled with their signal.

Redo Network Numbering [6-12] This command is not generally needed but may be useful if you suspect that the network information is incorrect.

Logical Effort [9-12]



This command does Logical Effort analysis, which determines the transistor ratios to use in digital schematic components in order to get optimal circuit speed.

Analyze Cell This command examines the current cell and annotates all schematic gates with fanout information.

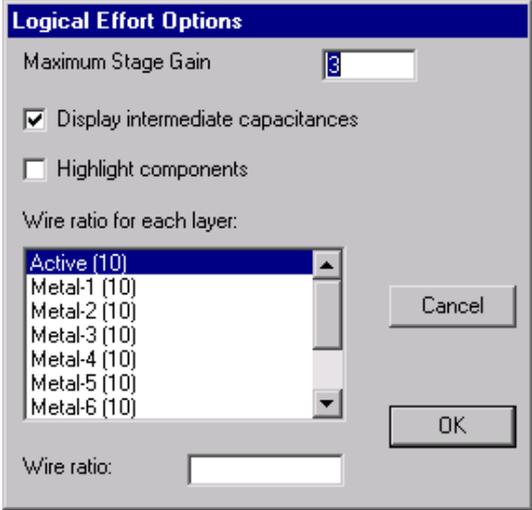


Analyze Path This command examines the circuitry between the two highlighted components and annotates the gates with fanout information.

New Arc Load This command creates a special "load" symbol that has capacitance and can connect to the circuit to declare a load there. The capacitance value can be changed by double-clicking and typing a new value.

Set Node Effort... This command allows you to set an overriding logical effort value on the currently selected node. This value can be changed by double-clicking and typing a new value.

Logical Effort Options... This command provides options for controlling the Logical Effort tool. The Maximum Stage Gain is used in the **Analyze Cell** command. The "Display intermediate capacitances" enables the display of capacitance values on arcs in the circuit. The "Highlight components" causes the tool to highlight the active components that it is analyzing.



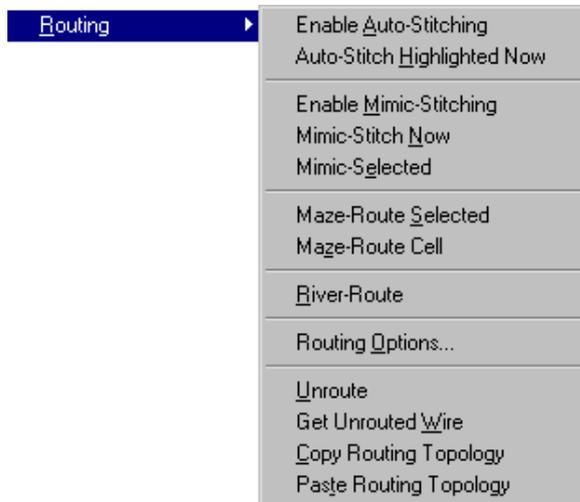
Finally, the dialog lets you set the wire ratio for each arc (a value used in load computation).

Estimate Delays This command examines each network in the circuit and calculates load factors. It is not generally useful, and applies to Logical Effort calculations.

Show Network Loads This command lists every network in the current cell, showing the wire length, load, and other information.

Analyze Network This command shows a detailed analysis of the currently selected network, including the area and perimeter information for each layer, as well as load information.





This submenu controls a number of wire routing facilities.

Enable Auto–Stitching This command instructs the router to watch all subsequent layout activity and to place arcs wherever touching nodes create implicit connections. It is useful to issue this command before generating arrays, because the array may produce many implicit connections that this router will make explicit. The menu entry changes to **Stop Auto–Stitching** to disable the function.

Auto–Stitch Highlighted Now This does auto–stitching only in the currently highlighted area. The highlighted area is defined as the bounding rectangle of everything that is highlighted. A more precise way of defining a highlighted area is to use the *rectangle select* button to drag a rectangle on the screen.

Enable Mimic–Stitching This command instructs the router to watch all subsequent layout activity and to automatically create other arcs in similar locations whenever you create one by hand. The menu entry changes to **Disable Mimic–Stitching** to disable this function.

Mimic–Stitching Now This command instructs the router to mimic the last arc that was created. It is not necessary for the Mimic Stitcher to be enabled.

Mimic–Selected This command instructs the router to mimic the selected arc. It is not necessary for the Mimic Stitcher to be enabled.

Maze–Route Selected This command runs the maze router in the selected area. All occurrences of the Unrouted wire will be replaced with real geometry.

Maze–Route Cell This command runs the maze router in the current cell. All occurrences of the Unrouted wire will be replaced with real geometry.

River–Route This command runs the river–router in the current cell. All occurrences of the Unrouted wire will be replaced with real

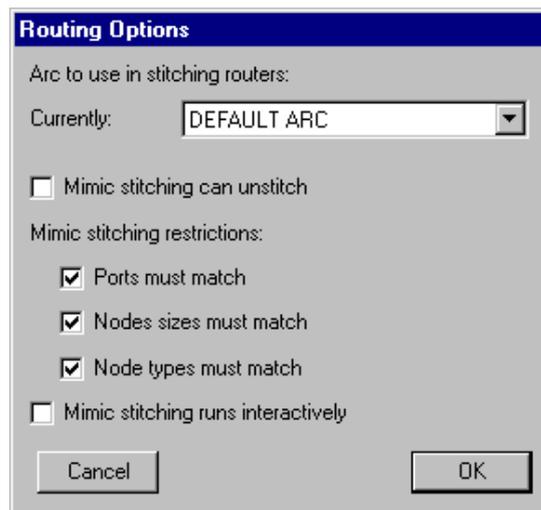


geometry.

Routing Options...

This command provides a dialog for control of the stitching routers.

For the Auto-Stitcher, you can select the type of arc to use (the default is to automatically determine the arc type to use from the ports).



For the Mimic-Stitcher, you can instruct it to mimic wire deletions as well as creations. The Mimic-Stitcher can also be instructed to relax its rules for mimicing (by default, arcs are mimiced if they run between the same ports on other nodes; however you can request that mimicing happens between any other ports that are the same distance apart as the original arc).

Unroute

This command takes the currently selected network(s) and converts them to unrouted wires. After this command, you can maze-route or river-route the unrouted wires.

Get Unrouted Wire

The Unrouted wire of the Generic technology is used to define routing requirements (see [Section 7-9](#)). This command selects the Generic arc so that subsequent wiring commands will use it. This is necessary in order to do maze and river-routing.

Copy Routing Topology

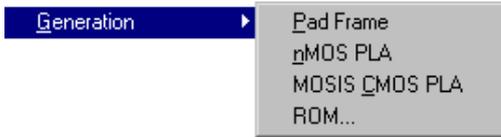
This command remembers the network of connections in the current cell for subsequent creation in another cell (specified with **Paste Routing Topology**).

Paste Routing Topology

This command examines the current cell and compares its network to the one that was copied (with **Copy Routing Topology**). Where there are missing connections in this cell, the command creates Unrouted arcs to connect them.

Generation





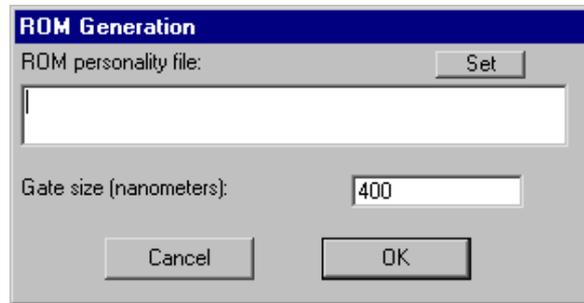
This command provides a pad frame generator and two PLA generators. All will run faster if the design-rule checker is turned off first.

Pad Frame [9-8] This command prompts for a disk file that describes the placement of pads around a core cell. The file includes information about the library that contains the pads and also the connection between the pads and ports on the core cell.

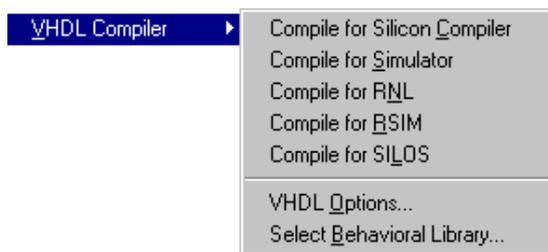
nMOS PLA [9-7] This command prompts for a personality table and generates nMOS layout, complete with power and clocking. See the description of the PLA generator for a sample personality table.

MOSIS CMOS PLA [9-7] This command prompts for two personality tables: the AND and the OR tables. It also offers options about the location of inputs and outputs. See the description of the PLA generator for a sample CMOS personality table.

ROM... [9-7] This command prompts for a personality table and generates a ROM. The code is implemented in Java, so this command is only active if you have installed Java into Electric.



VHDL Compiler [9-10]



This submenu provides direct control of the VHDL compiler, which translates VHDL textual descriptions into netlists. Besides controlling which format netlist is generated, it is also possible to determine whether the netlist of the VHDL is to be stored in memory (in a cell) or on disk.

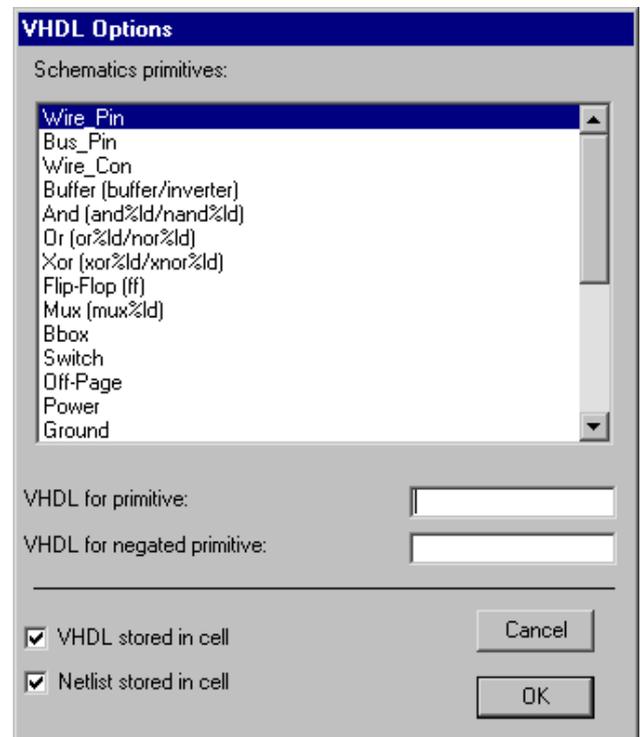
Compile for Silicon Compiler This command causes the VHDL in the current cell to be compiled into a netlist for the silicon compiler. If the current cell is not a VHDL view, the VHDL view is used. If VHDL disk files are being used instead of cells, the file "XXX.vhdl" is read, where XXX is the cell name of the current cell. If netlists are being written to disk, the file "XXX.sci" is written.

Compile for This command causes the VHDL in the current cell to be compiled into a netlist



- Simulation** for simulation. If the current cell is not a VHDL view, the VHDL view is used. If VHDL disk files are being used instead of cells, the file "XXX.vhdl" is read, where XXX is the cell name of the current cell. If netlists are being written to disk, the file "XXX.net" is written.
- Compile for RNL** This command causes the VHDL in the current cell to be compiled into a RNL simulator netlist. If the current cell is not a VHDL view, the VHDL view is used. If VHDL disk files are being used instead of cells, the file "XXX.vhdl" is read, where XXX is the cell name of the current cell. If netlists are being written to disk, the file "XXX.net" is written.
- Compile for RSIM** This command causes the VHDL in the current cell to be compiled into a RSIM simulator netlist. If the current cell is not a VHDL view, the VHDL view is used. If VHDL disk files are being used instead of cells, the file "XXX.vhdl" is read, where XXX is the cell name of the current cell. If netlists are being written to disk, the file "XXX.net" is written.
- Compile for SILOS** This command causes the VHDL in the current cell to be compiled into a SILOS simulator netlist. If the current cell is not a VHDL view, the VHDL view is used. If VHDL disk files are being used instead of cells, the file "XXX.vhdl" is read, where XXX is the cell name of the current cell. If netlists are being written to disk, the file "XXX.sil" is written.

VHDL Options... This command provides options for controlling whether the VHDL or the Netlist are stored in memory or on disk. Besides the compile subcommands in this menu, the state of these switches also affects the **Make VHDL View** command of the **View** menu, the **Simulate...** subcommand of the **Simulation (Built-in)** command of the **Tools** menu, and the **Get Network for Current Cell** subcommand of the **Silicon Compiler** command of the **Tools** menu.

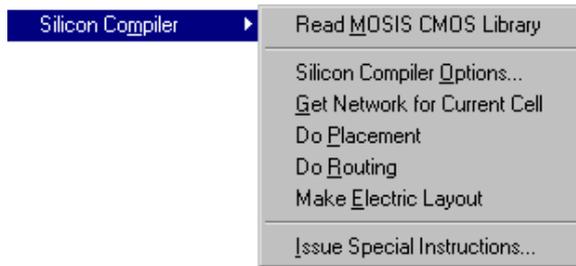


This command also controls the VHDL that is generated from schematics nodes. Each schematics node is shown, along with its regular and negated VHDL symbol (the use of "%d" is replaced by the number of inputs on the gate).

Select Behavioral Library... When compiling for simulation, behavioral models will be included if they are found in the current library. This command allows an alternate library to be searched for the models. Note that each model can be found in the "netlist-als-format" view of an appropriately named cell.



Silicon Compiler [9–9]



This submenu is an extensive system for placing and routing standard cell libraries from a structural VHDL description. Simply run each command in sequence: select a library, set options, obtain a netlist, place, route, and make Electric layout.

Read MOSIS CMOS Library

This command requests that the MOSIS CMOS standard cell library be used. This cell library is not guaranteed to be correct and exists only for illustration purposes. See the "Silicon Compiler" section of Chapter 9 for a description of the cells in this library.

Silicon Compiler Options...

This command presents a dialog that allows the setting of various parameters for the silicon compilation process.

Get Network for Current Cell

This command gets a netlist for the current cell. If the current cell is not a netlist, and the netlist associated with this cell is missing or out of date, the VHDL Compiler will be used to create a netlist. If the current cell is not VHDL, and the VHDL associated with this cell is missing or out of date, the VHDL will be generated from a schematic.

Do Placement

This command computes the placement of standard cells.

Do Routing

This command computes the routing among the placed standard cells.

Make Electric Layout

This command generates final circuitry from the computed placement and routing. The design-rule checker is turned off during this step.

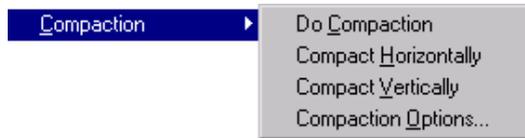
Issue Special Instructions...

This command allows you to communicate directly with the



Silicon Compiler. Only those familiar with the system should do this (the other commands in this submenu handle standard functions without the need to know how the compiler works).

Compaction [9–11]



This submenu controls the layout compactor.

Do Compaction

This command compacts the layout in the current window to design-rule distances, using single-axis compaction. It alternates horizontal and vertical compaction until no additional space can be saved. Compaction is done downward and to the left.

Compact Horizontally

This command instructs the compactor to compact the current cell one time in the horizontal direction.

Compact Vertically

This command instructs the compactor to compact the current cell one time in the vertical direction.

Compaction Options...

This command presents a dialog that allows you to tell the compactor to spread the circuit where it is too close for the design rules. You can also request information about the compaction process.



List Tools [9–1]

This command lists all of the tools, showing which ones are active.

Language Interpreter [11–1]





There are language interpreters in Electric: TCL, LISP, and Java. The interpreters can be activated with the subcommands here. Once activated, you communicate with them in the messages window. Note that, because of copyright restrictions, the LISP interpreter is not part of the standard GNU distribution and must be obtained separately from [Static Free Software](#). The TCL and Java interpreters must also be obtained separately. See the installation instructions for UNIX, [Section 1-3](#), Macintosh, [Section 1-4](#), and Windows, [Section 1-5](#).

