6

A Schematic Editor and Netlist Extractor in Java

by

David M. Liebson

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 17, 1997

Author____
Department of Electrical Engineering and Computer Science
May 17, 1997

Certified by____
Anantha Chandrakasan
Thesis Supervisor

Accepted by____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Schematic Editor and Netlist Extractor in Java

by

David M. Liebson

Submitted to the
Department of Electrical Engineering and Computer Science

May 17, 1997

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

In this thesis, I implemented a hierarchical schematic editor with netlist extraction capability for both Verilog and SPICE formats. This tool, WebTop, allows the rapid graphical development of electronic circuits. WebTop can interface with other CAD tools available on the Internet for circuit simulation and power estimation. WebTop can interact with distributed online libraries of cells to support collaborative design. WebTop was implemented completely in the Java programming language for maximum platform-independence.

Thesis Supervisor: Anantha Chandrakasan
Title: Analog Devices Career Development
Assistant Professor of Electrical Engineering

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Anantha Chandrakasan, for giving me the opportunity to work on this project.

My extreme gratitude to Debashis Saha, for his invaluable contributions of ideas, code, time, and raw debugging power! Debashis provided all of the Verilog support, most of the Web-oriented features, and the original design for hierarchical structures.

Thanks to Jennifer Mead, Danielle Coffing, and my entire family for moral support and constant encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

As use of the Internet becomes widespread, the potential for remote collaboration on design projects increases. However, the existing software tools at this time do not allow the user to use the Internet to maximum advantage. This thesis proposes to create part of a larger framework to enable the Internet-based design, simulation, and fabrication of integrated circuits and systems. Specifically, this thesis will involve the construction of a schematic editor and netlist extractor using Sun Microsystems' Java programming language [2]. This tool, known as WebTop, allows the user to draw a circuit schematic (referred to as a cell throughout this document) in a fully graphical, drag-and-drop interface and then to extract a netlist file suitable for simulation with various industry-standard circuit simulators, such as SPICE or Verilog. WebTop supports hierarchical structures: any given cell may be composed of other, lower-level cells. At the lowest level of abstraction, a large library of commonly used components is built in to WebTop to facilitate the creation of more complex cells. In addition to the built-in library of standard electronic components, additional component types may be created by the user and accessed either from local disk or a distributed network of World Wide Web (WWW) based object repositories (WebTop servers). WebTop is a Java applet that can theoretically be run under any Java environment. As of this writing, security restrictions in most available Java implementations prevent full functionality of WebTop. Full functionality is supported under Sun Microsystems' appletviewer program[1]; other

[1] Sun's appletviewer is available from the URL http://java.sun.com/products/jdk. WebTop was developed and tested using JDK version 1.02.

browsers such as Netscape Navigator[2] or Microsoft Internet Explorer[3] may have difficulties with operations involving local disk access or network access.

1.1 Web-based CAD

Now that a brief introduction to WebTop has been given, it is best to step back and examine its context in a wider field. Why is it useful to have a schematic editor that can work with the Web? First, collaboration in design becomes easier. If a common set of cells can be placed on an Internet server, geographically distant people can have easy, instant access to each other's work. If every user of WebTop makes the circuits (cells) that they create available publicly, then designs become progressively easier, as more complex cells become available. This public availability of cells can reduce redundant effort; perhaps a cell you need has already been designed by someone else. Also, the availability of other circuits online may can provide reference when designing new circuits. Existing circuits can suggest new methods or techniques to be used in other projects.

Another benefit of Internet integration is the ability to connect to other tools that are already available on the Internet. Such tools include circuit simulators and power dissipation estimators. Connection to simulation tools is interesting, because circuit simulators are often expensive and computationally intensive. Because the schematic

[2] Netscape Navigator is available from the URL http://www.netscape.com. WebTop was tested with Netscape Navigator version 3.01.

[3] Microsoft Internet Explorer is available from the URL http://www.microsoft.com/products/inteprod.asp. WebTop was tested with Microsoft Internet Explorer version 3.0.

editor is not itself computationally intensive, low end machines can be used for the design work, while a dedicated, powerful server processes simulations. Cost can be reduced, as each server can simulate circuits submitted by many users, reducing the need to purchase multiple copies of the software.

Connection to power estimators is also extremely interesting, because power dissipation estimation is a difficult, often overlooked aspect of the design process. Use of tools specifically designed for power analysis can greatly reduce simulation time needed.

In the future, all sorts of CAD tools will be available on the Internet. Soon, it should be possible to complete the entire design process, from drawing the schematic to ordering the fabrication, all from a machine connected to the Internet.

1.2 Available Web-based Tools

As noted previously, some CAD tools are already available on the Internet.

One such tool is WebSpice, developed by Debashis Saha in conjunction with the WebTop project. WebSpice accepts a SPICE netlist as input, processes the netlist, and returns the results to the user.

In the power estimation category, there are at least two WWW-based tools existing. One is Pythia[7], developed by Thucydides Xanthopoulos. Pythia accepts a Verilog netlist, and estimates the power dissipation of the design.

Another power estimator is PowerPlay[3], by David Lidsky. PowerPlay works more on a higher, more conceptual level than Pythia does. Rather than accepting a netlist, PowerPlay asks questions about capacitance, frequency, area, and other circuit parameters. Not all of these parameters are required; PowerPlay does the best it can with the information it is given. WebTop does not interface with PowerPlay at this time; however, if in the future WebTop is expanded to allow for a more conceptual, "top-down", concept-oriented design style, PowerPlay integration may be a useful future feature.

Another tool worth mentioning is the applet upon which WebTop is based. WebTop is based upon a program called "Digital Simulator," released into the public domain by Iwan van Rienen [6]. This program is written in Java, and implements a basic schematic editor along with a simple digital simulator. However, this applet does not have all of the features desired for the schematic editor. The program is not extensible, in that no additional components can be defined without recompiling the entire program. No hierarchical features are present. Also, netlist extraction is not supported. Although simulation features are present, and useful for design verification, complete simulation is not possible with the built in features. Although the "Digital Simulator" lacks some features, the user interface is excellent. Thus, the interface portions of Digital Simulator have been retained, and the extra features (such as hierarchy and netlist extraction support) have been added to produce WebTop. Digital Simulator has been used as a starting point by other research groups working on Web-based tools. Notably, the Web-Based Electronic Design (WELD) project at the University of California, Berkeley [5], modified Digital Simulator. The WELD project's modifications enable Digital Simulator to interact with their Synopsys design compiler[4]. However, the WELD project's modifications do not add all of the features desired.

Thus, there are tools available to simulate and perform useful analysis of electronic circuits. However, no tool exists to permit the efficient creation of input for these other tools. WebTop, which is an extended version of the original Digital Simulator applet[5], has been developed to meet this need.

2. OVERVIEW OF WEBTOP

This section is intended to give a general overview of what WebTop does, how WebTop was developed, and how WebTop is structured internally.

2.1 Purpose of WebTop

WebTop is a hierarchical schematic editor that supports netlist extraction into various formats. What exactly does this mean?

First and foremost, WebTop is a schematic editor. The fundamental purpose of WebTop is to allow the user to generate nice-looking schematics on the screen in a straightforward manner. That is to say, WebTop functions as a drawing program for schematics. The figure below shows an example of a circuit drawn with WebTop. Cells

[4] The modified version of Digital Simulator is available from the URL http://yoyodyne.eecs.berkeley.edu/DigSim/DigSim.html. Documentation of the modifications made to the Digital Simulator applet by the WELD project can be found at the URL http://yoyodyne.eecs.berkeley.edu/DigSim/readme.html.

[5] WebTop was developed from the original distribution of the Digital Simulator applet. The WELD project's modifications are not incorporated into WebTop.

can be placed, moved, and deleted with ease. The usual cut, copy, and paste operations are supported as well, for repetitive structures.

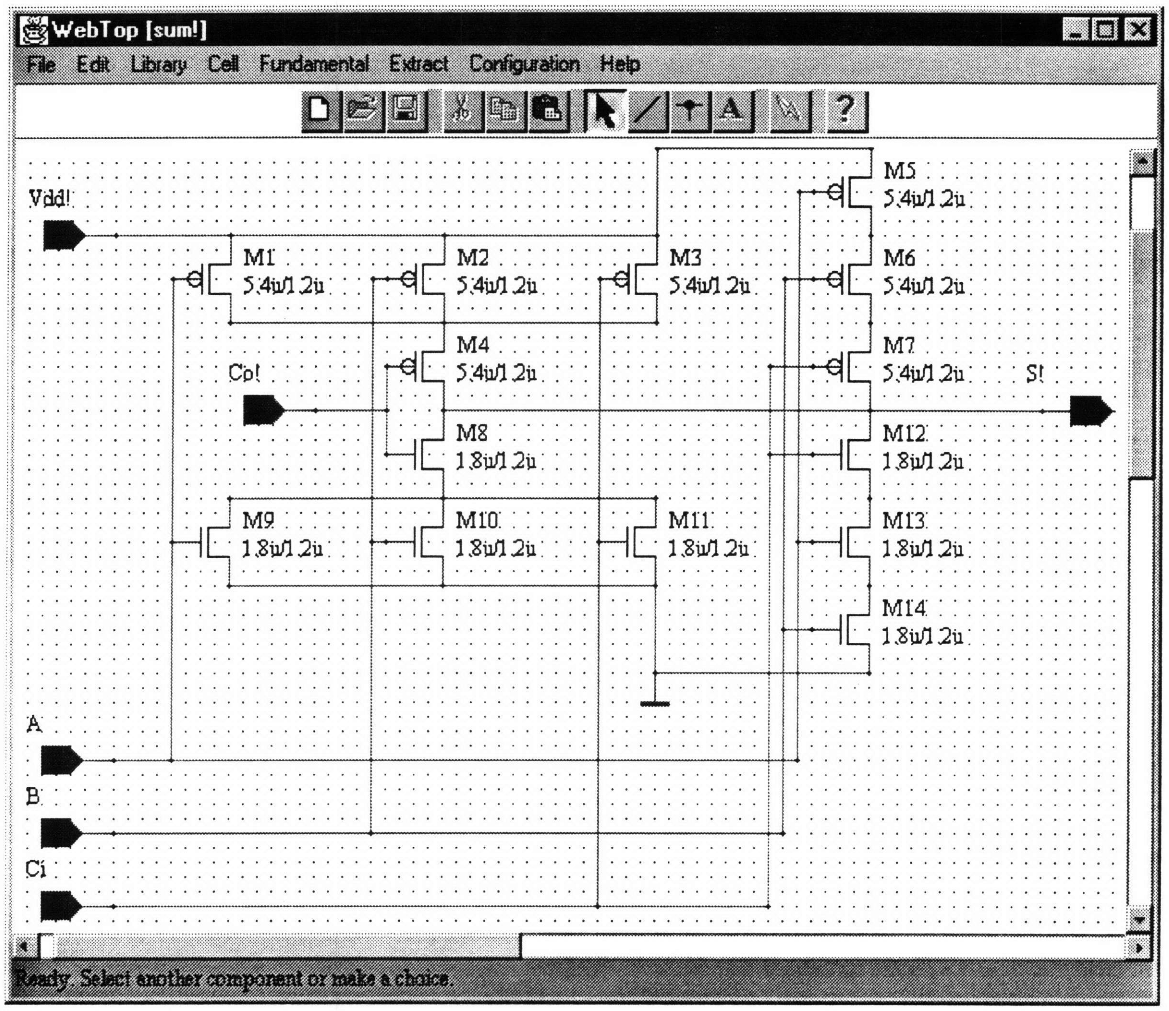


Figure 1: Example of a circuit design in WebTop

However, if the schematic has many repetitive structures, hierarchy is probably a better option than repeated cutting and pasting. WebTop's hierarchical features allow a collection of cells to be made into another cell. Only the specified nodes are brought out of the cell for external connection. Thus, complexity can be easily managed. Figure 2 shows the circuit of Figure 1 represented in this "black box" format. Such conversion from the full circuit representation of Figure 1 to the "black box" representation of Figure

2 is automatic. In the terms used by WebTop, Figure 1 shows the schematic view of the sum cell and Figure 2 shows the symbol view.

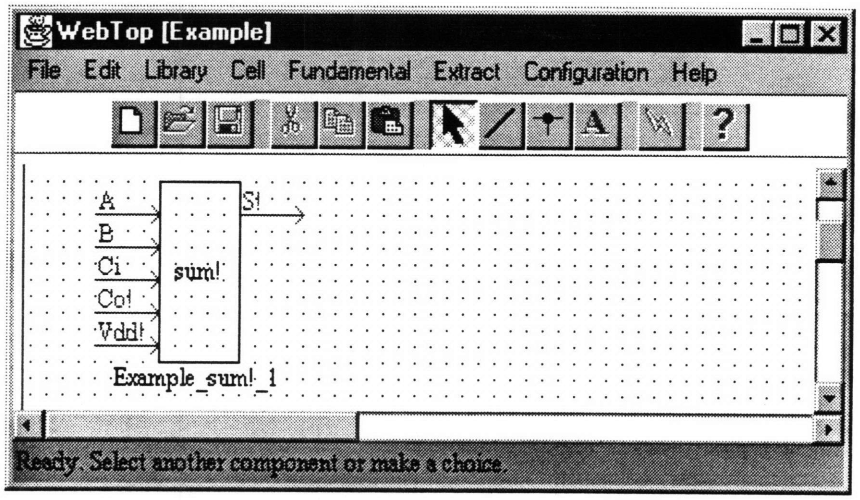

WebTop [Example]
File Edit Library Cell Fundamental Extract Configuration Help
A
B
Ci
Co!
Vdd!
S!
sum!
Example_sum!_1
Ready. Select another component or make a choice.

Figure 2: Example of a circuit shown as a "black box"

User defined SchematicCells can be saved to local disk, or accessed from networked WebTop servers. Thus, users can easily create and access libraries of useful cells.

Of course, this would all be useless if you couldn't actually do anything with these cells. WebTop supports netlist extraction in both SPICE and Verilog formats. That is, once you draw your cell, you can automatically generate a netlist suitable for SPICE or Verilog simulation[6]. Instead of drawing the circuit out on paper, labeling the nodes, and generating the SPICE netlist by hand, you get the netlist automatically. For example, for the example circuit from Figures 1 and 2, we get the following SPICE netlist:

```
* WEBTOP Spice Netlister *
* Copyright: Massachusetts Inst. Of Tech *
```

[6] The Pythia power estimator supported by WebTop has restrictions on the types of Verilog input it can process. The Verilog output produced by WebTop is target towards processing by the Pythia power estimator.

```
.MODEL NMOS NMOS LEVEL=2 LD=0.15U TOX=200E-10
+ NSUB=5.37E15 VTO=.74 KP=8E-5 GAMMA=0.54
+ PHI=0.6 U0=656 UEXP=0.157 UCRIT=31444
+ DELTA=2.34 VMAX=55261 XJ=0.25U LAMBDA=0.037
+ NFS=1E12 NEFF=1.001 NSS=1E11 TPG=1.0 RSH=70
+ CGDO=4.3E-10 CGSO=4.3E-10 CJ=0.0003 MJ=0.66
+ CJSW=8E-10 MJSW=0.24 PB=0.58

.MODEL PMOS PMOS LEVEL=2 LD=0.15U TOX=200E-10
+ NSUB=4.33E15 VTO=-0.74 KP=2.7E-5 GAMMA=0.58
+ PHI=0.6 U0=262 UEXP=0.324 UCRIT=65720
+ DELTA=1.79 VMAX=25694 XJ=0.25U LAMBDA=0.061
+ NFS=1E12 NEFF=1.001 NSS=1E11 TPG=-1.0 RSH=121
+ CGDO=4.3E-10 CGSO=4.3E-10 CJ=0.0005 MJ=0.51
+ CJSW=1.35E-10 MJSW=0.24 PB=0.64

*** TopLevel Cell Instancesum! ***
Xsum! A B Ci Co! Vdd! S!  sum!
*** End TopLevel Instance ***

*** TopLevel Sub-ckt: sum! ***
.SUBCKT  sum! A B Ci Co! Vdd! S!
MM1 Vdd! A _node_1 Vdd! PMOS w=5.4u l=1.2u
MM2 Vdd! B _node_1 Vdd! PMOS w=5.4u l=1.2u
MM3 Vdd! Ci _node_1 Vdd! PMOS w=5.4u l=1.2u
MM4 _node_1 Co! S! Vdd! PMOS w=5.4u l=1.2u
MM5 Vdd! A _node_2 Vdd! PMOS w=5.4u l=1.2u
MM6 _node_2 B _node_3 Vdd! PMOS w=5.4u l=1.2u
MM7 _node_3 Ci S! Vdd! PMOS w=5.4u l=1.2u
MM8 S! Co! _node_4 0 NMOS w=1.8u l=1.2u
MM9 _node_4 A 0 0 NMOS w=1.8u l=1.2u
MM10 _node_4 B 0 0 NMOS w=1.8u l=1.2u
MM11 _node_4 Ci 0 0 NMOS w=1.8u l=1.2u
MM12 S! Ci _node_5 0 NMOS w=1.8u l=1.2u
MM13 _node_5 A _node_6 0 NMOS w=1.8u l=1.2u
MM14 _node_6 B 0 0 NMOS w=1.8u l=1.2u
.ENDS
*** End TopLevel Sub-ckt ***
.end
* -- Extraction Completed --*
```

This netlist is generated completely automatically. No knowledge of SPICE is necessary to create the basic netlist. Once the netlist is created, simulation control statements can be added, and the SPICE file submitted to the WebTop server for simulation.

This is a very basic introduction to WebTop's functionality. Appendix A presents an exhaustive list of WebTop's features.

2.2 Implementation of WebTop

WebTop was developed in Sun Microsystems' Java programming language. Java was chosen due to its flexibility and platform-independence. Furthermore, some previous work in schematic editors was available in the public domain. WebTop is based on a public domain applet called "Digital Simulator" (DigSim). Although WebTop looks at first glance very much like the original, unmodified DigSim, the appearance is misleading. Although the interface looks similar, there is little similarity in source code between the two applications.

As of this writing, Sun is supporting two different versions of the Java specification. The older version, JDK1.02 is widely supported by third-party web browsers. The newer version, JDK1.1 supports many useful features not included in JDK1.02. However, as of this writing, no third-party web browsers support the JDK1.1 specification. In order to test the functionality of the application, we had to be able to run it. Thus, WebTop has been developed using the JDK1.02 specification.

One new JDK1.1 feature was used in WebTop, because patches to JDK1.02 are available to implement the features[7]. These are the remote-method invocation features

[7] The patches to add RMI features to JDK1.02 are available from the URL http://chatsubo.javasoft.com. These patches are pre-beta unsupported software but have performed adequately in testing.

(RMI), used by WebTop for efficient file storage and loading. The utility of the RMI features was sufficient to warrant its use, even with the inconvenience of installing patches.

According to Sun's documentation, JDK1.02 applications should continue to run in JDK1.1 environments. Testing indicates that this is untrue for most non-trivial Java program. Most problems result from Sun's replacement of methods with similar, but different calls. Sun's documentation specifies that such deprecated methods should continue to work for at least one JDK release after their deprecation. In practice, such methods are often removed in the next JDK release.

These problems between version of the JDK specification will eventually require that WebTop be converted to JDK1.1 when JDK1.1-compliant web browsers become standard. Problems in porting the WebTop code to JDK1.1 should be minimal. The JDK1.1 compiler will produce a warning when it finds outdated code. Sun provides a reference to find the updated versions of deprecated methods (reference).

WebTop has been tested under Microsoft Windows NT 4.0 (Intel processor) and Sun Solaris 5.4. On the Intel platform, WebTop has been tested using Sun's appletviewer, Netscape Navigator, and Microsoft Internet Explorer. On the Sun machine, WebTop was tested under appletviewer and Netscape Navigator. As WebTop is written entirely in Java, it should in theory run properly on other platforms. However, during development, many compatibility problems were detected between platforms, and even between different browsers on the same platform. Because most of the software related to Java, as well as the Java specification itself is evolving rapidly at this point in time, testing is essential to verify the WebTop is functional under a given environment before a large-scale design effort is started. Unfortunately, there is no guarantee that WebTop will be compatible with the future versions of software that WebTop works with now, due to the rapid evolution of the Java specification.

2.3 Overall Structure of the Code

WebTop is a fairly large application. It consists of approximately 13,000 lines of code, spread across 55 different class files. Approximately half of the class files are descended from code present in the original Digital Simulator applet; the other half are completely new. The files descended from the original Digital Simulator have for the most part been heavily modified. While attacking this amount of code may seem intimidating at first, it is fortunately not as bad as it looks. The classes in WebTop break down into three major categories: data structures, graphical user interface components, and PrimitiveCells. Many classes include features of both the data structure and GUI component categories, but in general a class falls more naturally into one of the categories than the other. The applet class itself (WebTop), due to its top-level nature, is best treated on its own, as it combines the previous categories, as well as having distinct characteristics of its own.

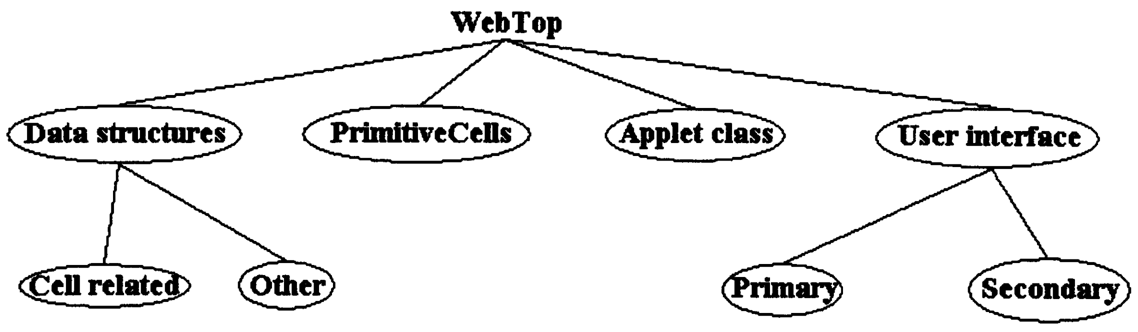


Figure 3: WebTop overall structure

Data structure classes are further classified by what their purpose is. One large group of classes implements the hierarchical cell structure used by WebTop. The base class is Cell, and other classes such as PrimitiveCell and SchematicCell descend from it. Other data classes used to implement the cell structure include IOPin, which specifies an interface from a cell to the rest of the world. Not all of the data structure classes are directly related to the cell structure. The Schematic class represents an arbitrary collection of cells, and is used for tracking groups of cells. For example, one Schematic structure might contain all of the cells on the screen, one schematic may contain the results of a cut, copy, or paste option, and so on. In general, the data structure classes are used to store, manage, and manipulate data that is used by other parts of the applet.

Graphical User Interface classes are responsible for managing the interaction between the user and the applet. Anything you see displayed on the screen is an instance of a GUI component class. Often, the GUI component classes display the data contained within a data structure class to the user. GUI component classes can be further divided into two main groups: primary and secondary. Primary GUI component are the GUI components that the user spends the most time interacting with. Primary GUI components would include SchematicPanel, ColorManager, and others. Secondary GUI components tend to be transient in nature – they appear only long enough to display some information, or to request some information from the user. Secondary GUI components may also be "helpers" for primary GUI components. Thus, in the following detailed discussion of WebTop internals, secondary GUI component classes are discussed along with the primary GUI component they are associated with. Secondary GUI component classes in WebTop include ComponentBox, TextView, and so on.

PrimitiveCell classes implement the basic building-block cells. Circuit components such as resistors, transistors, capacitors, voltage sources, and so on are PrimitiveCells. Each different PrimitiveCell that WebTop supports has its own class, because each component needs different initialization, has a different drawing routine, interacts with netlist extraction differently, and so on. Approximately half of the class files in WebTop are PrimitiveCell classes.

3. WEBTOP IN DETAIL

With that brief overview of WebTop's internal organization, it is possible to explore WebTop's inner workings in a more detailed manner. This section is intended to provide some understanding of how the different class files of WebTop interact to produce a working application, and perhaps to explain some of the more subtle points of some of the functions. It is not intended to be a complete guide to WebTop; for that, the source code must be consulted. The source code is commented, so that looking at the source code will be most beneficial. The best way to learn about how WebTop works is to read this thesis, use WebTop, and then read the source code.

In this section, only the most important members and methods of each class are discussed. If a member or method is not mentioned in this section, it is either of minor importance, or straightforward to understand by reading the source code. A reference guide to the class files of WebTop is provided in Appendix C.

3.1 Data Structures

As mentioned before, WebTop contains various data structures that are used to storing and manipulating critical data. Most of these data structures are related to the hierarchical cell structure; some are not.

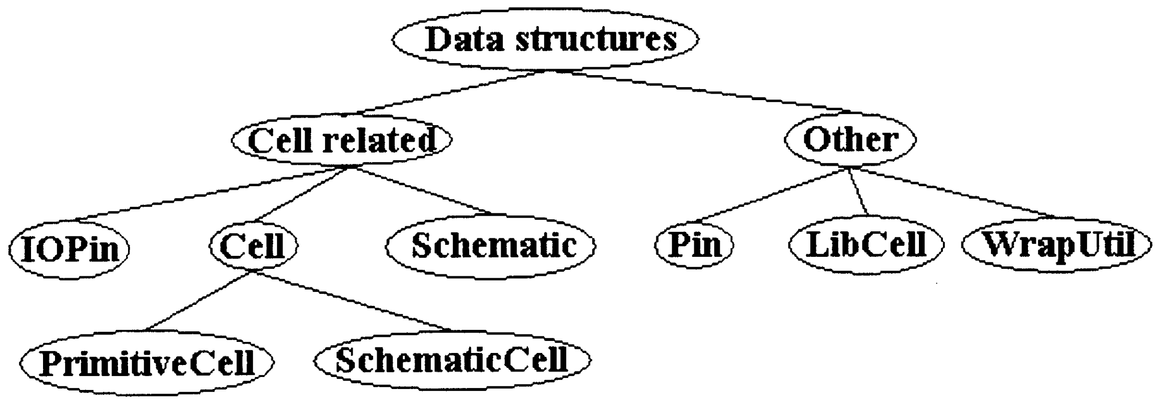


Figure 4: WebTop data structure classes

3.1.1 Cell related data structures

These classes are generally concerned with representing cells, groups of cells, or portions of cells. This subset of classes consists of: IOPin, Cell, PrimitiveCell, SchematicCell, and Schematic.

IOPin

An IOPin is a portion of a cell. Specifically, it represents a connection between cells. All PrimitiveCells have IOPins; most SchematicCells have IOPins as well.

Because IOPins are the only way to connect between cells, any cell that is to be re-used in other designs must have IOPins. The only type of cell that would not have IOPins is the absolute top-level cell of a design.

The IOPin structure, in its role as a connection between cells, must keep track of several important pieces of data. Thus, each member of the IOPin is discussed separately.

The first member, int type, represents the kind of pin that the IOPin is. These types are defined statically and finally in IOPin.java. In this version, there are input pins, output pins, and input/output pins. The type distinction matters only when drawing the pin. An input-type IOPin represents an input to the cell from the rest of the world, and is drawn from the active end to the passive end. with an arrow at the passive end. An output-type IOPin represents an output from the cell to the rest of the world, and is drawn from the passive end to the active end, with an arrow at the active end. An input/output-type IOPin is a simple line segment connecting the active end and the passive end, and is drawn with no arrow.

As mentioned above, each IOPin has an active end and a passive end. The active end is the end that wires can connect to. The passive end cannot be connected to, and is intended to be placed adjacent to the main body of its parent cell.

The active end of the IOPin is tracked by the member Point IOPinPos. This point is specified relative to the origin of the parent component.

The passive end of the IOPin is implicitly tracked by the member Dimension IOPinDim. IOPinDim represents an (x,y) offset from the active end of the IOPin to the passive end.

The member String pin_name tracks the "real" name of the IOPin. This name never changes, no matter how the cell might be connected. This name is intended to be descriptive of the pin's function: "Input A", "Vdd", "Output" and so on.

The member String lexical_name tracks the name of the circuit node that the IOPin is connected to. This name is set when extracting a netlist, and can of course change depending on how the IOPin is connected.

The member boolean showtext controls whether or not the pin_name is displayed when the pin is drawn. For a PrimitiveCell, showtext is generally false. For a SchematicCell, showtext is true.

The final member Vector ConnComps is used only when extracting a netlist. It is used to track which other cells this IOPin is connected to.

Cell

The Cell class represents the basic cell, which is the base of all hierarchical structures. This class is abstract; you will never actually encounter a generic cell object. Often, descendants of the Cell class are cast to type Cell to simplify processing of large numbers of cell. The various types of cells are rather similar, so most of the functions can go into the parent class.

The member Hashtable Parameters contains various parameters of the cell, such as name, value, and so on. These parameters can affect how the Cell is drawn, and particularly how the Cell is represented in extracted netlists. The keys of this hashtable are Strings representing the name of the parameter. The associated key is a String containing the value of the parameter.

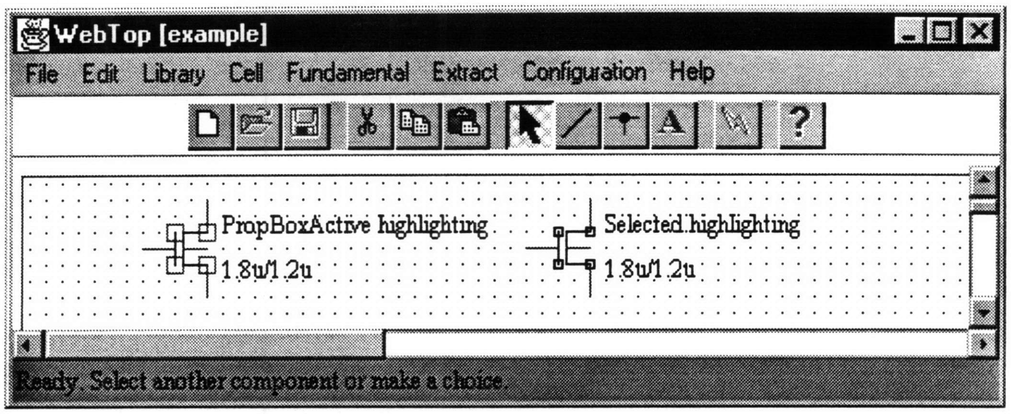
The member Vector pins contains a list of all IOPins associated with this Cell.

The member String views[] contains an array of views (Symbol, Schematic, Spice, Verilog, etc.) set for this cell. Views do not have any particular meaning for PrimitiveCells, but they do matter for SchematicCells. The views are included in the generic Cell class to simplify processing of large numbers of cells.

The members Point Pos and Dimension Dim track the position and size of the Cell. Pos specifies the upper left-hand corner of the Cell relative to the origin of the SchematicPanel. Dim contains the offset of the lower right-hand corner of the cell, relative to the upper left-hand corner.

The members Point HitBox and Dimension HitBoxSize perform a similar function to the previously mentioned Pos and Dim members. However, they track aspects of the HitBox, which is the area that is used to determine if the Cell has been clicked upon. The HitBox should always be contained entirely within the area specified by Pos and Dim.

The members boolean Selected and boolean PropBoxActive control different types of highlighting that maybe applied to a Cell. Selected-style highlighting is used when the user has clicked on the component. It consists of small squares drawn around the corners of the Cell's HitBox. PropBoxActive-style highlighting is applied to a Cell when a PropertyBox associated with this Cell has the user input focus. It is similar to Selected-style highlighting; however, the squares for PropBoxActive-style highlighting are larger than Selected-style highlighting. It is important to note that a given Cell may have no highlighting, one type, or both types simultaneously. Figure 5 below depicts both types of highlighting.


WebTop [example]
File Edit Library Cell Fundamental Extract Configuration Help
PropBoxActive highlighting
1.8u/1.2u
Selected highlighting
1.8u/1.2u

Ready. Select another component or make a choice.

Figure 5: Cell highlighting

PrimitiveCell

The PrimitiveCell class is a descendent of the Cell class. It is used to represent the built-in Cell types that come with a release of WebTop. These PrimitiveCells are the basic electronic components that arbitrary, more complex components can be built up from. PrimitiveCells can not be supplied or modified by the end user of WebTop.

The PrimitiveCell has no members not present in the generic Cell type. Some extra constructors are provided, and the abstract methods of the Cell class are properly overridden.

SchematicCell

The SchematicCell class is a descendent of the Cell class. It is used to represent a user-defined Cell. Basically, a SchematicCell is a Cell that can contain other cells. These cells can be of the PrimitiveCell or SchematicCell type.

The SchematicCell class has three members not present in the generic Cell class. The member Vector subcells contains the other cells that are contained within this SchematicCell. The members int totalGridX and int totalGridY indicate the size of the pin grid for this cell. The size of the pin must be tracked in case the Expand Horizontal or Expand Vertical commands are used while editing a cell; if a cell larger than the default 100x100 size were to be loaded without re-sizing the pin grid, errors would result.

The SchematicCell class has a few interesting methods that are not a part of the Cell class. These methods deal with the extra complexity of the subcells.

The GetSubCell() method is used to extract the essential data from a SchematicCell. It produces a "stripped-down" version of the SchematicCell. The resulting SchematicCell preserves the name and the pins of the original SchematicCell.

The fixup() method is used to properly determine how to draw the SchematicCell. Because SchematicCells can have different name lengths and different numbers of pins, the SchematicCell class must be able to account for this variation. The fixup() method sets the Dim, HitBox, and HitBoxSize members of the SchematicCell, and the IOPinPos and IOPinDim members of each associated IOPin.

The dumpCell() and parseCell() methods are used for saving and loading WebTop-format files. The dumpCell() method produces a String representing the SchematicCell in the WebTop file format described in Appendix B of this document. The parseCell() method performs the inverse operation of producing a SchematicCell from a WebTop-format String. In this version, neither method is particularly robust with respect to error handling. The parseCell() method, in particular, would benefit from a real parsing scheme, rather than using the java.util.StringTokenizer routines.

Schematic

The Schematic class represents an arbitrary collection of cells. The Schematic class is used extensively throughout the GUI portions of the WebTop code. The Schematic structure was present in the original Digital Simulator package, and has continued into WebTop. The Schematic could be partially replaced in some portions of the user interface code with proper references to the Vector subcells of the SchematicCell class, but often the code is easier to understand in terms of the Schematic class. The Schematic class is essential to the Cut, Copy, and Paste operations. The Schematic class is also used to track which components in the current cell are selected. Even though the Schematic class is used primarily for user interface functions, it is classified along with the cell-related data structures because of its role as an arbitrary collection of cells.

The member Vector Components tracks each cell present in the Schematic. It is similar, but not identical, to the Vector subcells of the SchematicCell. The Vector subcells of the SchematicCell contains all of the subcells of the SchematicCell, whereas the Vector Components of the Schematic can be empty, contain some subset of the Cells in the current Cell, contain Cells not present in the current Cell, and so on.

The member boolean Modified keeps track of whether or not modifications have been made since the current Cell was last saved. This is used for user interface purposes. By checking the state of this member, WebTop can determine whether or not to ask for confirmation before discarding the current Cell.

The method PasteSchematic() is rather complicated. Cutting and pasting is generally complicated in WebTop, because there are many special cases that can occur. Furthermore, duplicate names must be avoided, aliasing between objects must be avoided, and proper consistency between all data structures must be maintained. This method is sufficiently complicated that a quick guide to what it does is appropriate. The first for-loop of this method iterates over all of the Components to find the minimum x and y values of the component positions. These values are used to determine where to place the Cells that are to be pasted. The next for-loop also iterates over all of the Components. A copy of each component Cell is made (the actual component itself cannot be used, or aliasing problems would result). Next, this new copy is given a new name, such that its name does not conflict with the name of any other Cell present. Because there are special cases of cells requiring special behavior, the generation of new names accounts for the bulk of this method's code. Once the new name is generated, the new Cell is added to the current Cell. Finally, the current Cell is checked to make sure that no cycles have been introduced.

The method RemoveSameElements() removes the cells in a schematic from that schematic, as well as from the current Cell. This method is called when a number of Cells are selected, and the Cut command is performed. The selected cells must be removed from the current cell, as well as from the schematic containing all currently selected components.

3.1.2 Other data structures

Not all of the data structure classes in WebTop are used to manage Cells. Some data structures are used to maintain other information.

Pin

The Pin class represents a connection point for components. The schematic editor maintains a pin grid; that is a two-dimensional array of Pin objects. Schematics are build upon this grid. Each point of the grid is a single Pin object.

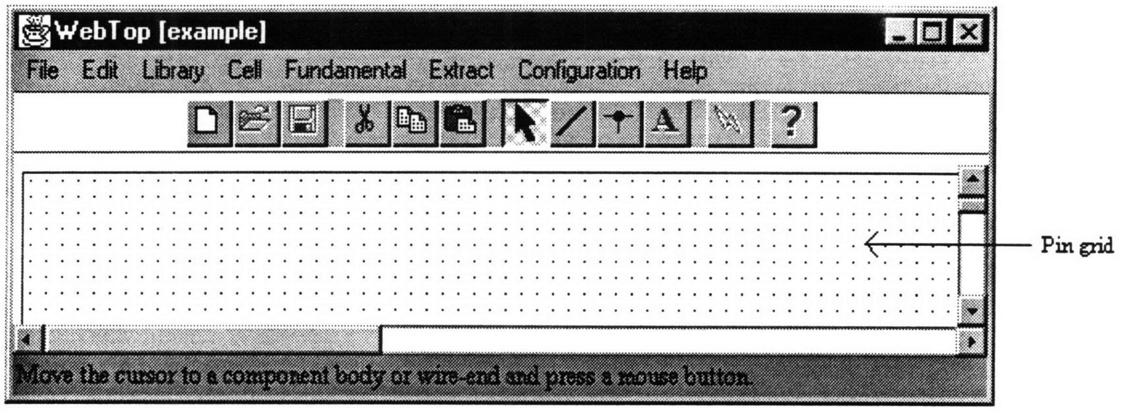

WebTop [example]
File Edit Library Cell Fundamental Extract Configuration Help
Pin grid
Move the cursor to a component body or wire-end and press a mouse button.

Figure 6: Pin grid

The member Vector Components keeps track of the Cells that are connected to the Pin. This information is used when extracting a netlist. Care must be taken to update this pin information every time a Cell is added, deleted, or moved.

LibCell

The LibCell structure is a condensed form of a Cell used by the library manager. It extracts only the information about a cell that is needed by the library manager; the type of the Cell (currently, all Cells in the library manager are of type SchematicCell), the name of the Cell, and the views associated with the Cell. This information is used to generate the lists seen in the library manager itself.

WrapUtil

WrapUtil is not actually data structure; rather, it is a collection of useful methods that don't really fit anywhere else. Because these methods are used for the manipulation of data, WrapUtil falls somewhat into this general category of classes.

The methods in WrapUtil are used to prepare data or headers to send to network servers. The methods allow specification of delimiters and various parameters used to prepare the data.

3.2 GUI Components

As previously mentioned, many of the class files in WebTop implement features of the user interface

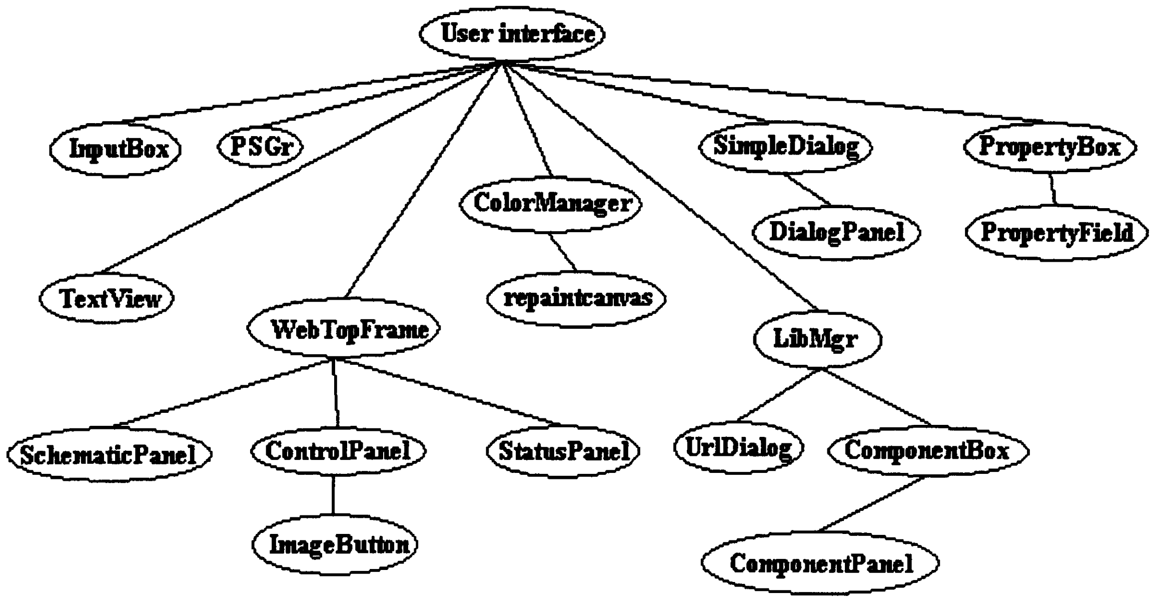


Figure 7: User interface classes

3.2.1 ControlPanel and ImageButton

The ControlPanel is a part of the schematic editor window. It is panel of buttons that may be clicked to execute commands. This panel is provided as a fast way to access certain common commands without having to select them from the menus. Buttons are provided for the New cell, Cut, Copy, Paste, Pointer mode, Wire mode, and Junction mode commands. The ControlPanel itself is composed of a collection of ImageButton objects.

Buttons may be in one of several different states. Buttons can be enabled, disabled, pressed and selected. In each case, the button has a slightly different appearance. All of the different button possibilities are loaded from one master image, 'images/allbuttons.gif'. This master image is loaded into the Image CopyImage member of the ControlPanel class. The code checks the state of the button, and then copies the appropriate button image from the master CopyImage into the appropriate location in the Image ImageBuffer member. The Image ImageBuffer member contains the graphics of all of the buttons of the ControlPanel, and is used to paint the ControlPanel. Methods are provided to set the state of each button. This is necessary because some buttons, such as Cut, Copy, and Paste, may be enabled or disabled at different times during the operation of WebTop.

The ControlPanel implements the mouseMove(), mouseEnter(), and mouseExit() convenience methods in order to provide a simple sort of help to the user. The mouseMove() method checks which button the mouse pointer is in. If the mouse pointer is in an enabled button, the schematic editor's status message is updated with a message indicating which button the mouse is on. This functionality is supported in case the user forgets which icon is associated with each function.

The ImageButton class is a single button of the ControlPanel. Most of the actual functionality is in the ControlPanel class. ImageButton provides the state variables for each button (selected, pressed, enabled) and methods for manipulating the state of each button. The draw() method of each ImageButton calculates the proper range of the CopyImage that should be copies to show this ImageButton in the proper state.

3.2.2 SchematicPanel

The SchematicPanel is part of the schematic editor window. The SchematicPanel displays whatever cell is currently being worked on, and handles most of the user interaction.

The SchematicPanel class contains many members for tracking the position of the window, the last component selected, and so on. Most of these members are self-explanatory. The members int GridXOffset and int GridYOffset are used to help determine which part of the SchematicPanel is actually visible on the screen. Because the SchematicPanel itself is larger than the window that contains it, scrollbars are necessary. GridXOffset and GridYOffset represent the coordinates, in gridpoints, of the upper left-hand corner of the screen. The SchematicPanel also contains two Schematic objects as members. These members, Schematic SelectSchematic and Schematic CopySchematic, are used for cut, copy, and paste operations. The SelectSchematic contains all cells that are currently selected. When a cell is selected, it is added to the SelectSchematic; when a cell is deselected, it is removed from the SelectSchematic. The CopySchematic implements the buffer in cut, copy and paste operations.

When the Cut command is issued, the SelectSchematic is copied to the CopySchematic. The cells in the SelectSchematic are removed from the current cell, and the SelectSchematic is emptied. When the Copy command is issued, the SelectSchematic is copied to the CopySchematic, and retained. When the Paste command is issued, a copy of the CopySchematic is made, and added to the current cell.

The SchematicPanel is responsible for much of the user interaction in WebTop. Thus, a large amount of the code is dedicated to handling user input. For the most part, convenience methods such as mouseDown() and mouseUp() have been used for clarity.

Although JDK1.02 does not explicitly support keyboard shortcuts for commands, such functionality can be supported by handling keypress events. Unfortunately, different Java environments differ in handling of keyboard events. The observed differences have been in which GUI component receives the events. For example, on Microsoft Windows platforms, the SchematicPanel will receive any keyboard events generated while the SchematicPanel has the input focus. However, on Sun Solaris platforms, keyboard events are generated in the WebTopFrame. The practical consequence of this difference is that any critical keyboard shortcuts implemented by handling keypress events must be handled in both SchematicPanel and WebTopFrame.

For efficiency reasons, the paint() and update() methods of SchematicPanel are a bit more complicated than absolutely necessary. First, it is important to distinguish between the purposes of the paint() and update() methods. The paint() method just paints on the screen; it assumes that nothing has changed state. The paint() method is called by the Java system when the component has been covered by another window, moved, or otherwise affected by other aspects of system operation. The update() method actually does most of the work of drawing the SchematicPanel properly. When something about what is to be displayed changes, it is update() that should be called. This is why the Java repaint() method calls update(), not paint(). The actual graphics displayed by the paint() method are held in an Image structure called ImageBuffer. The paint() method checks that nothing actually needs to be updated, and then simply draws the existing ImageBuffer. The update() method is more complicated, as it must actually build the ImageBuffer. To build the ImageBuffer, the update() method starts with an image of the grid, draws a border around it, draws all of the cells on top of it, then adds other things that may or not be present, such as wires in the process of being drawn, a SelectBox, junctions, and so on.

3.2.3 StatusPanel

The StatusPanel is part of the schematic editor window. It is a small panel at the bottom of the schematic editor window used to display messages to the user. These messages provide information about what mode the editor is in (Wire, Pointer, Junction), help about the ControlPanel, and so on.

3.2.4 WebTopFrame

The WebTopFrame is the schematic editor window. It contains the ControlPanel, SchematicPanel, and StatusPanel, and the schematic editor's menus. The schematic editor's menus comprise most of WebTopFrame's members. The member Vector AvailableComponents contains a list of PrimitiveCells available to WebTop, and is used to simplify some of the menu handling code. The member Vector MenuItemsToDisable is an artifact of the original "Digital Simulator" applet. It is used to disable menus that could possibly perform actions at inappropriate times. In the original applet, MenuItemsToDisable would be used while a simulation was running. In the current version of WebTop, menus are always available. However, the MenuItemsToDisable structure has been maintained in the event that such functionality would be useful in the future.

The WebTopFrame is also responsible for handling the events of the SimpleDialog class. The events of the SimpleDialog class are handled here, instead of in the SimpleDialog itself, so that callbacks to the top-level WebTop applet class can be made. Such calls could be made by adding a reference to the applet to the SimpleDialog class, but this arrangement allows the same functionality in a less confusing manner.

Most of the code of the WebTopFrame is to handle the menus. In the handleEvent() method, the label of the activated menu item is checked, and the appropriate action taken. Most menu commands call routines in the top-level applet class WebTop.

3.2.5 LibMgr, ComponentBox, ComponentPanel, and UrlDialog

The LibMgr is responsible for maintaining collections of cells. LibMgr also is responsible for netlist extraction in both Verilog and SPICE formats, and communication with Internet-based tools. Most members of LibMgr are used to maintain the lists of libraries, cells, and views available. Some members are used to handle netlist extraction.

Many of the methods of LibMgr are used to manipulate cells; addCell(), deleteCell(), getCell() and so on. This type of method is straightforward.

Another group of methods is used to connect the LibMgr and the top-level WebTop applet class. These are the methods that perform major actions of the LibMgr that affect the schematic editor, such as adding a cell, editing a cell, and so on.

The addToSchematic() method is called to add a new SchematicCell from the LibMgr to the current cell in the SchematicPanel. The cell to be added is copied from the LibMgr and given a proper name. Then, the Cell is checked to ensure that its addition to the current cell will not cause cycles in the extracted netlist. Next, the SchematicCell.fixup() method is called. This method must be called so that the new SchematicCell will draw properly. The information needed to properly draw a given SchematicCell is not necessarily saved with the SchematicCell, so the information must be regenerated any time the SchematicCell is used.. The new SchematicCell is then ready to be added into the SchematicPanel's current cell.

Many methods are provided for loading and saving cells in a variety of formats. These methods are bulky, but straightforward.

The final grouping of methods is used for the netlist extraction process. These methods represent each Cell in the circuit as a sub-circuit, and then define every sub-circuit necessary with the proper details.

LibMgr has a few small helper classes associated with it. These classes are small interface components used by the LibMgr to implement some of its functions. Two of these classes, ComponentBox and ComponentPanel, are used to display the symbol view of a Cell in the library. ComponentPanel is an extension of Panel with the paint method overridden to draw the symbol view of a Cell. ComponentBox is an extension of Frame that contains the ComponentPanel. The Frame class cannot be used by itself, because the Frame class does not handle WINDOW_DESTROY events, and we want to be able to close the window properly. The third helper class, UrlDialog, allows the user to enter parameters necessary to load a cell from a URL.

3.2.6 ColorManager and repaintcanvas

The ColorManager is used to configure the colors used to display various elements of WebTop's interface. The ColorManager provides a list of elements that colors can be set for, scrollbars to set the color, and an area that displays a sample of the current color.

The only interesting aspect of the ColorManager's implementation is the code to handle the scrollbars. This code requires extra care, because it seems that every Java environment WebTop has been tested in handles scrollbars in a slightly different manner. Scrollbars can generate five different events that are of interest: SCROLL_LINE_UP, SCROLL_LINE_DOWN, SCROLL_PAGE_UP, SCROLL_PAGE_DOWN, and SCROLL_ABSOLUTE. The line events are generated when the scrollbar is moved a single unit in either direction (by clicking on the arrow). The page events are generated by clicking in the scrollbar itself. The absolute event is generated by clicking on the scrollbar's slider, positioning it, and releasing the mouse. Figure 8 shows a scrollbar and identifies the event that would be generated by clicking on specific locations.

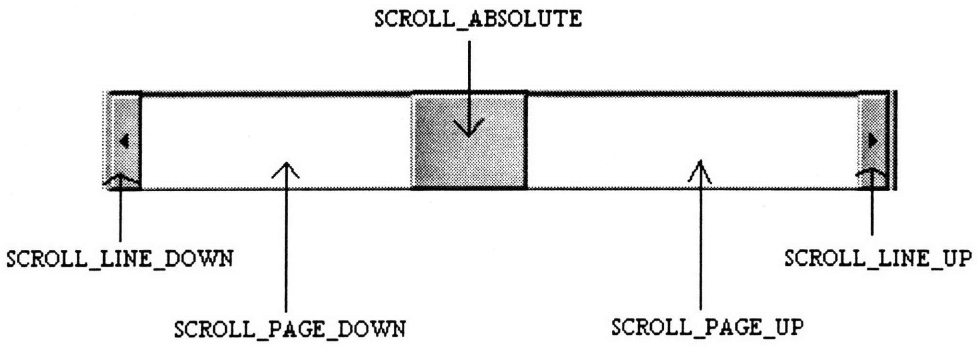


Figure 8: Scrollbar and events

Although one would expect the Scrollbar itself to handle these various events without intervention by the user, this is not the case on the Microsoft Windows platform. For each type of event, the value of the scrollbar must be explicitly set to the value contained within the argument of the event that is generated, or the scrollbar does not update at all. The behavior of the scrollbars differs remarkably between different Java environments. Scrollbars in Microsoft Internet Explorer behave properly. Netscape Navigator doesn't generate SCROLL_ABSOLUTE events; instead, a series of SCROLL_LINE events are generated as the slider is positioned. This is a disaster when a large window must be repainted many times on a slow machine. Sun's appletviewer will hang if scrollbar events are generated in rapid succession. Any attempt to generate a

SCROLL_ABSOLUTE event will hang, as will holding down the mouse button in a SCROLL_PAGE region. No solution or source for these bugs could be located.

3.2.7 PropertyBox and PropertyField

The PropertyBox is used to set the Parameters of a cell. The PropertyBox can appear when the cell is first added, or by the user double-clicking on the cell in the schematic editor window. The implementation of PropertyBox is for the most part straightforward.

Several aspects of the code are worth noting. First, some of the Parameters of the Cell may be read-only (such as instanceName). Therefore, a test must be performed when a list item is selected. If the selected list item should be read-only, the PropertyField must be set to its non-editable state. The other interesting method of the PropertyBox is the method HighlightCell(). This method is used to adjust the SchematicPanel's viewing area so that the component associated with this PropertyBox is visible on screen. This method calculates the range of pin grid indices that are visible on screen, tests if the Pos member of the associated Cell is within that range, and adjusts the range if necessary.

The PropertyField is a simple extension of TextField. The PropertyField captures the TextField's ACTION_EVENT, which is generated when the user presses return in the TextField. The updateValue() method of the PropertyBox is called when the ACTION_EVENT is generated. The GOT_FOCUS event is also handled so that the PropBoxActive highlighting of the associated Cell does not disappear when manipulating the data in the PropertyField.

The repaintcanvas class is used to implement the color sample of the ColorManager. The repaintcanvas is an extension of canvas with a paint method that draws the sample area.

3.2.8 InputBox

The InputBox is a simple extension of Dialog used only to enter a name for a new Cell. That is, the InputBox is only called a part of the sequence performed when the "New Cell" command is issued. The InputBox is a modal dialog; it will block the input to other interface objects while it is onscreen. This is desirable in this case because we don't want the user editing the new Cell until it is completely set up; It is easier to set the name and related parameters of the new Cell when it is still in a fresh, known state.

3.2.9 TextView

The TextView is used to show the SPICE and Verilog views of the Cell. It consists of a frame with an optionally editable TextArea for displaying the view, and a few buttons for control purposes. The implementation of the TextView is straightforward.

3.2.10 SimpleDialog and DialogPanel

The SimpleDialog is used throughout to pop up a message to the user. The dialog's text, buttons, and optional graphics are all specified in the call to the SimpleDialog constructor. The events of the SimpleDialog are handled by the WebTopFrame class.

The DialogPanel class is used to contain all of the data of the DialogPanel, including the buttons, captions, and optional graphics. This class was implemented separately to keep the interface to the SimpleDialog clean.

The DialogPanel must take the caption String supplied to the SimpleDialog (which is a sing String, possibly containing '/n' characters) and split it into multiple Strings (one for each line of the caption) that Graphics.drawString() can handle.

The DialogPanel is also responsible for loading and managing the optional image that may be displayed in the SimpleDialog. Graphics for 'warning' and 'error'/'stop' are provided for added emphasis.

3.2.11 PSGr

PSGr is a class used for outputting Adobe PostScript files[8]. Adobe PostScript is a popular page formatting language used in most printers [1],[4]. Thus PostScript is a logical output format, due to its wide support. PSGr extends the standard AWT.Graphics context, such that the usual Graphics methods can be used to generate PostScript output. This is indeed fortunate, because all of the existing draw() methods for components can be reused.

The version of PSGr used within WebTop is slightly modified to meet particular requirements. In particular, the regularly released version of PSGr does not support landscape printing, does not support scaling to fit a page, and does not support margins.

[8] PSGr is a part of a larger Java applet called Jlpr. PSGr was developed by Ernest Friedman-Hill and is included in WebTop under his copyright notice: PSGr is (C) 1996 Ernest Friedman-Hill and Sandia National Labs. Right to unrestricted personal and commercial use is granted if this acknowledgment is given on product or packing materials.

WebTop will automatically determine whether landscape or portrait mode is most appropriate. That is, the schematic will be oriented such that the longer dimension of the schematic is along the long edge of the page. Furthermore, WebTop will scale the schematic to fit on one page. This scaling is one way; the schematic will be reduced in size if necessary, but will not be expanded if the default size takes less than a full page. If the schematic is very large, the output may be too small to read. However, due to the hierarchical features of WebTop, a single schematic should rarely grow to such a size.

3.3 PrimitiveCell Classes

WebTop supports twenty-five different PrimitiveCell classes. Of these PrimitiveCell classes, only a few of them are particularly interesting from an implementation standpoint. Most of the PrimitiveCell classes are simple descendants of PrimitiveCell, simply overriding the necessary methods, but not implementing any additional methods.

Four of the PrimitiveCell classes warrant special treatment: Wire, isoInputPin, isoOutputPin, and isoInputOutputPin. The classes isoInputPin, isoOutputPin, and isoInputOutputPin are collectively referred to as "isoPin."

Wire must be implemented somewhat differently do to the completely different method by which wires are drawn and manipulated. Despite being lumped with the PrimitiveCell classes, a Wire is not in fact a PrimitiveCell! Nothing is particularly subtle about the workings of the Wire class.

The three different varieties of isoPin are special because they represent the IOPins in a SchematicCell's schematic view. When building a new SchematicCell, the various forms of isoPin are used to indicate a connection between the outside world to the cell. In the implementation, care must be taken to maintain consistency between the isoPin cells included in the schematic view of a SchematicCell and the SchematicCells pin structure. This special case handling appears primarily in Schematic, SchematicCell, and WebTop, not in the implementations of the isoPin classes themselves.

The PrimitiveCell classes may be divided up into categories based on what extraction formats they permit.

PrimitiveCells not supported by SPICE extraction, but supported by Verilog extraction are: AND, Buffer, Inverter, NAND, NOR, OR, XNOR, and XOR.

PrimitiveCells supported by SPICE extraction, but not supported by Verilog extraction are: C, E, G, I, L, NMOS4, PMOS4, R, V, and Vcc.

PrimitiveCells supported by both SPICE and Verilog extraction are: NMOS and PMOS.

PrimitiveCells not supported by either SPICE or Verilog extraction are: GND, Wire, isoInputPin, isoOutputPin, and isoInputOutputPin. These components should be filtered out by the netlist extraction code. Any attempt by the netlist extractor to include these components is a bug in WebTop, and will be reported as an error by the netlist extractor code.

3.5 Top-level WebTop Applet

The WebTop class is the basic, top-level class that contains everything else. Many of the most important members and methods of WebTop are included in the WebTop class itself to provide more global access. Most of the methods of WebTop are called from other classes, but are included in the WebTop class to take advantage of the local access to important data structures.

All of the main user interface structures are members of WebTop. These members are: WebTopFrame frame, ControlPanel MyControlPanel, StatusPanel MyStatusPanel, SchematicPanel MySchematicPanel, LibMgr libmgr, and ColorManager cm

The members Image ImageBuffer and Image GridImage are used for drawing purposes. Image ImageBuffer contains the actual graphics drawn to the screen. Image GridImage contains a image of grid points on the proper background color. The purpose of these images is discussed in the SchematicPanel section.

WebTop also contains data structures to track the contents of the current cell. The member Schematic MySchematic contains every cell that is part of the current cell. The member SchematicCell TopCell actually *is* the current cell. The member SchematicCell NewTopCell is used to temporarily hold a cell when an operation calling for discarding the current cell is performed.

WebTop also contains some data structure-type classes for managing important data. The member Pin PinGrid[][] is a two-dimensional array of Pin objects that keeps tracks of where components are connected and what they are connected to. Conceptually, the PinGrid[][] corresponds to the grid of points drawn in the SchematicPanel. The member Hashtable Comphash is used to generate names for new cells added to the current cell.

WebTop contains many other members to keep track of positions, sizes, and other important but straightforward features. Most of the methods are also straightforward and simple. However, some of them are not.

The UserWantsEditSchematic() method is called when the user wants to see the schematic view of a SchematicCell. This method first gets confirmation that the user really wants to discard the current cell. Next, the current cell is discarded. Data structures such as the PinGrid, GridImage, and Comphash are rebuild to be accurate for the new cell. Finally, the SchematicPanel is repainted so that the updates are visible.

The most complicated methods of WebTop are used for labeling circuit nodes automatically. Circuit nodes are labeled prior to netlist extraction, or when saving.

The UserWantsExtact() method is the most important method relating to node extraction. It is the only method in WebTop related to node labeling that is called directly; the other methods in WebTop related to node labeling are called by this method. First, the lexical_names of IOPins must be cleared. Next, each Pin in the PinGrid updates its list of connected components. Next, every node that is connected to some external source must be labeled. This includes any circuit node that is connected to an isoInputPin, isoOutputPin, isoInputOutputPin, or GND cell. If any cell is found to be connected to more than one external source, an error message is generated. After all of the external names are propagated throughout the circuit, any nodes that are not labeled are given names.

The AssignNames() method is actually responsible for naming nodes. The most difficult part of naming a node is to find everything that is connected to a given circuit node, because many components can be connected to a single circuit node by wires. It is also important to avoid infinite looping while determining what is part of each circuit node. This method accepts several parameters. The types of these parameters are Cell, IOPin, boolean, String, and boolean. The IOPin is the most important, as it reveals which IOPin we are attempting to label. The Cell is the cell that the IOPin is attached to. The String and booleans are used when forcing a certain name to propagate from another point, such as a GND cell or an isoInputPin. This method checks if the IOPin in question already has a name. If it does, an error has occurred and is reported. If not, the IOPin is named. Next, everything else connected to this component is named as well. This includes any other components that may be connected directly to this IOPin, as well as anything else connected to any wires connected to this IOPin. Wires are difficult to deal with, because they can connect to other wires, form loops, and so on. Thus, each time we encounter a wire, it is added to a Vector. This Vector should be checked every time a new wire is encountered, to make sure looping does not occur.

The method FindOtherEnd() is used when propagating a pin name down a wire. Two forms of this method, differing in the parameters they take, are used. Each version accepts parameters describing one end of the wire, and returns the Pin connected to the other end of the wire.

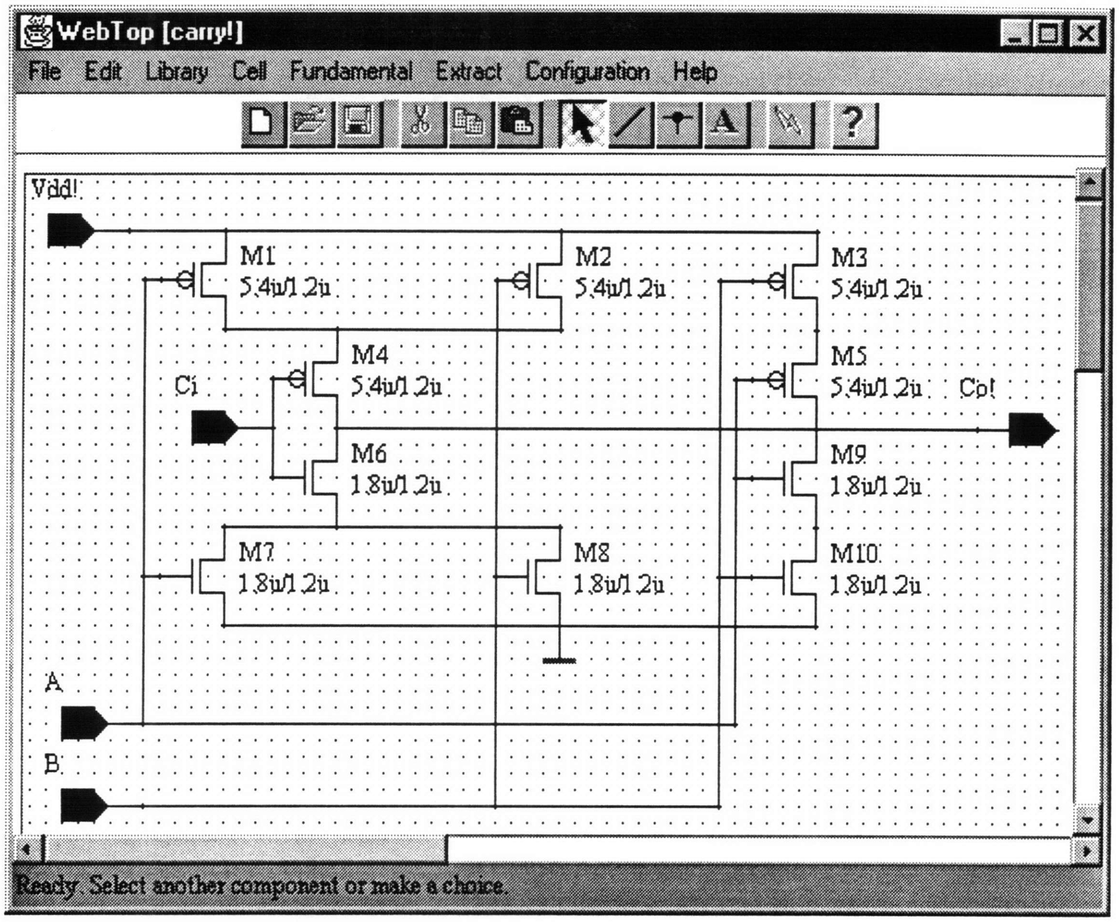
Finally, the method KeepGoing() is used to propagate names down wires. It is similar in function to the assign names; its function is to follow along any wire that is encountered and assign the proper name to any pins connected.

4. DESIGN EXAMPLE

At a number of points in this thesis, the hierarchical features of WebTop have been mentioned. This section will work through a example using several layers of hierarchy to properly demonstrate the hierarchical features.

For this example, a four bit ripple-carry adder will be constructed at the transistor level. The base adder design will be a simple static CMOS mirror adder. The hierarchical features of WebTop will be used to keep each schematic a reasonable size. To bit the ripple-carry adder, we can simply chain one bit adders together. To build a one bit adder, both the sum and carry signals must be generated. Each of these signals can be generated independently to keep the maximum Cell size down.

First, we build a cell to generate the carry signal. This cell takes input signals A, B, and Ci, as well as a power supply voltage Vdd!, to produce the output signal Co!



Ready. Select another component or make a choice.

Figure 9: Carry generator circuit

Next, we build a cell to generate the sum signal. This cell takes the same input signals as the previous cell. This cell also takes a Co! signal as input. S! is produced as an output

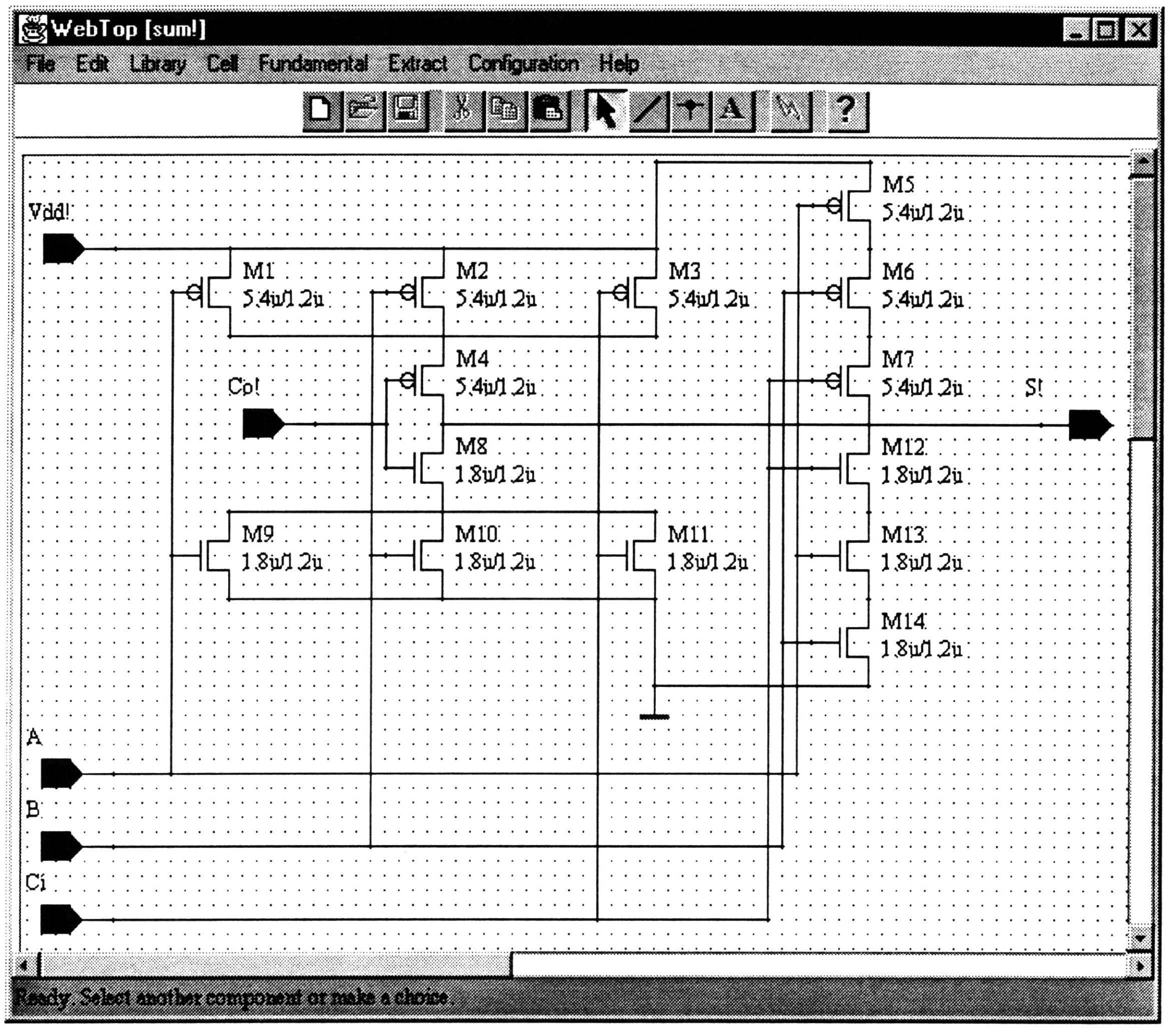


Figure 10: Sum generator circuit

The above circuits produce the complements of the actual sum and carry signals desired. So, an inverter cell is needed to produce the proper signal polarity.

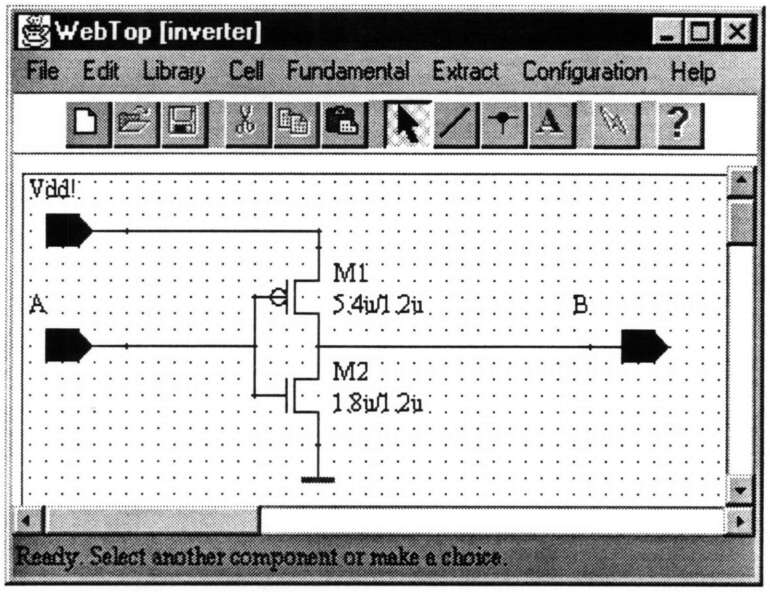

WebTop [inverter]
File Edit Library Cell Fundamental Extract Configuration Help
Vdd!
A
M1
5.4u/1.2u
B
M2
1.8u/1.2u
Ready. Select another component or make a choice.

Figure 11: Inverter circuit

Now, all the cells needed to produce a full adder are ready. To produce the full adder, just connect together the cells already made.

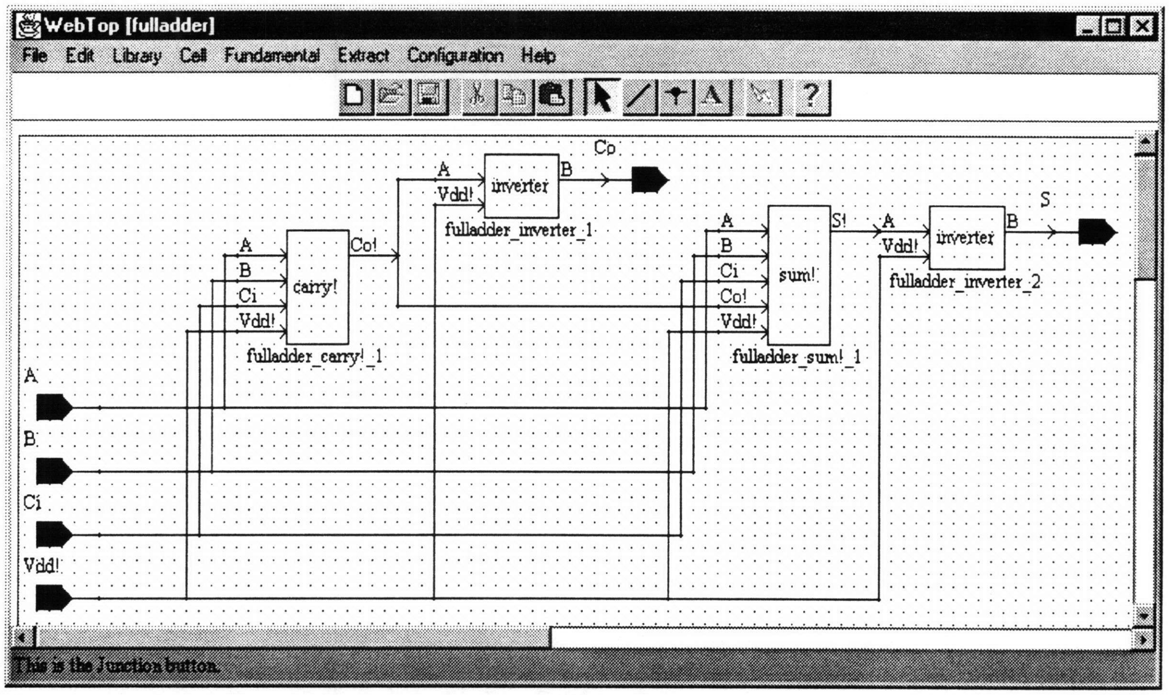


Figure 12: Full adder circuit

Now that a one bit adder exists, it's easy to chain four of them together to produce a four bit ripple-carry adder.

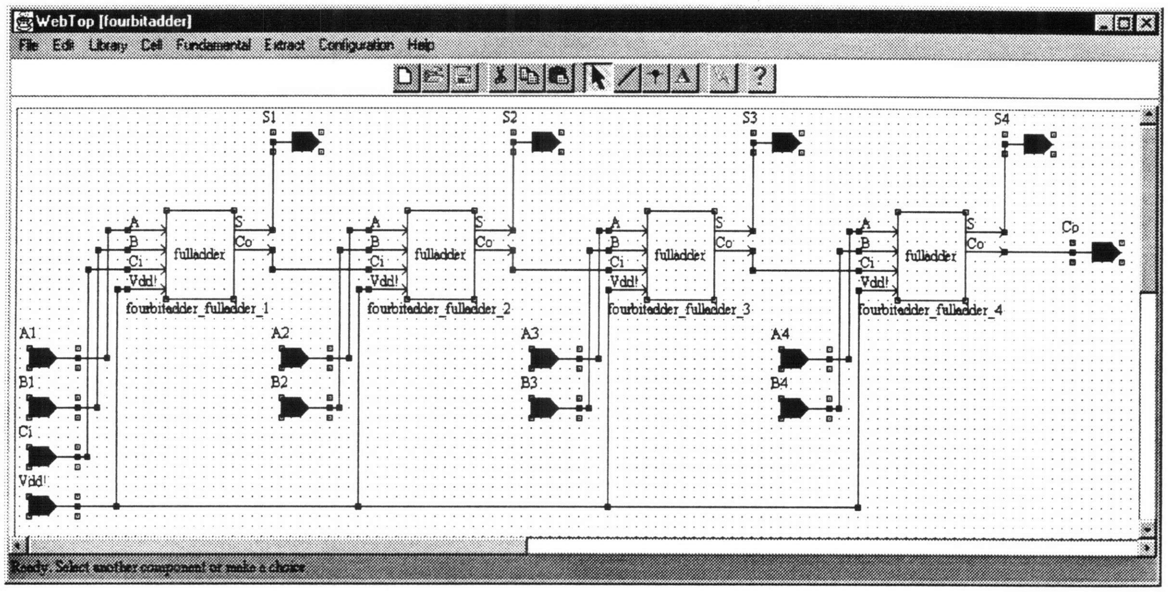


Figure 13: Four bit ripple-carry adder

Obviously, arbitrarily large adders could be built up by chaining these four bit adder cells together.

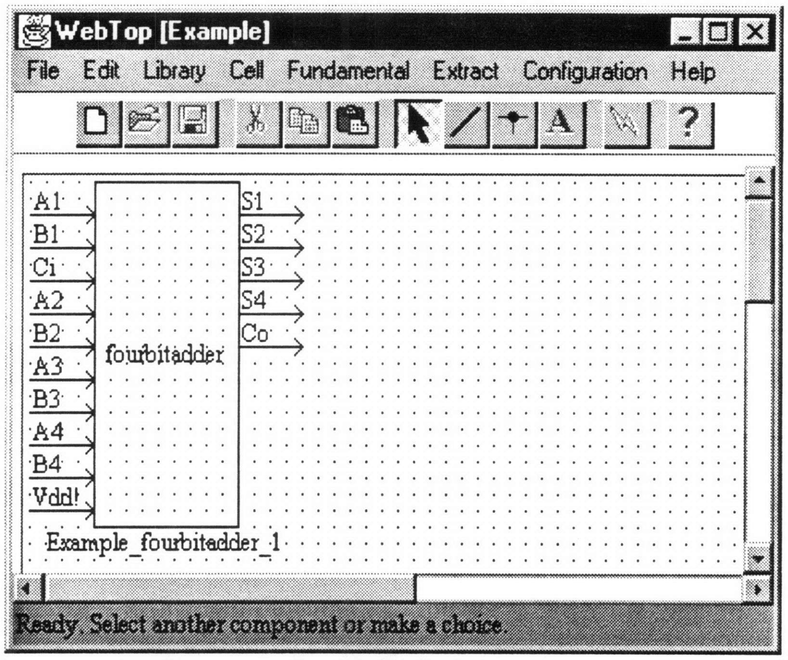

WebTop [Example]
File Edit Library Cell Fundamental Extract Configuration Help
A1
B1
Ci
A2
B2
A3
B3
A4
B4
Vdd!
fourbitadder
S1
S2
S3
S4
Co
Example_fourbitadder_1
Ready. Select another component or make a choice.

Figure 14: Four bit adder ready for use

This example, although simple, shows how powerful this hierarchy is. The simple box in Figure 14 above contains almost one hundred transistors – but is available for future designs at the single click of the mouse!

The SPICE netlist generated by the four bit ripple-carry adder pictured in Figure 14 is as follows. Note how each sub-circuit is only defined once, so increasing the size of the adder wouldn't increase the size of the resultant SPICE file by a large amount.

```
* WEBTOP Spice Netlister *
* Copyright: Massachusetts Inst. Of Tech *

.MODEL NMOS NMOS LEVEL=2 LD=0.15U TOX=200E-10
+ NSUB=5.37E15 VTO=.74 KP=8E-5 GAMMA=0.54
+ PHI=0.6 U0=656 UEXP=0.157 UCRIT=31444
+ DELTA=2.34 VMAX=55261 XJ=0.25U LAMBDA=0.037
+ NFS=1E12 NEFF=1.001 NSS=1E11 TPG=1.0 RSH=70
+ CGDO=4.3E-10 CGSO=4.3E-10 CJ=0.0003 MJ=0.66
+ CJSW=8E-10 MJSW=0.24 PB=0.58

.MODEL PMOS PMOS LEVEL=2 LD=0.15U TOX=200E-10
+ NSUB=4.33E15 VTO=-0.74 KP=2.7E-5 GAMMA=0.58
+ PHI=0.6 U0=262 UEXP=0.324 UCRIT=65720
+ DELTA=1.79 VMAX=25694 XJ=0.25U LAMBDA=0.061
+ NFS=1E12 NEFF=1.001 NSS=1E11 TPG=-1.0 RSH=121
+ CGDO=4.3E-10 CGSO=4.3E-10 CJ=0.0005 MJ=0.51
+ CJSW=1.35E-10 MJSW=0.24 PB=0.64

*** TopLevel Cell InstanceExample ***
XExample  Example
*** End TopLevel Instance ***

*** TopLevel Sub-ckt: Example ***
.SUBCKT  Example
XExample_fourbitadder_1 _node_1 _node_2 _node_3 _node_4 _node_5_node_6
_node_7 _node_8 _node_9 _node_10 _node_11 _node_12 _node_13 _node_14
_node_15  fourbitadder
.ENDS
*** End TopLevel Sub-ckt ***

*** Sub-Circuit fourbitadder
.SUBCKT  fourbitadder A1 B1 Ci A2 B2 A3 B3 A4 B4 Vdd! S1 S2 S3 S4 Co
```

```
Xfourbitadder_fulladder_1 A1 B1 Ci Vdd! S1 _node_1   fulladder
Xfourbitadder_fulladder_2 A2 B2 _node_1 Vdd! S2 _node_2   fulladder
Xfourbitadder_fulladder_3 A3 B3 _node_2 Vdd! S3 _node_3   fulladder
Xfourbitadder_fulladder_4 A4 B4 _node_3 Vdd! S4 Co   fulladder
.ENDS
*** End Sub-Circuit

*** Sub-Circuit fulladder
.SUBCKT   fulladder A B Ci Vdd! S Co
Xfulladder_sum!_1 A B Ci _node_1 Vdd! _node_2   sum!
Xfulladder_carry!_1 A B Ci Vdd! _node_1   carry!
Xfulladder_inverter_1 _node_1 Co Vdd!   inverter
Xfulladder_inverter_2 _node_2 S Vdd!   inverter
.ENDS
*** End Sub-Circuit

*** Sub-Circuit sum!
.SUBCKT   sum! A B Ci Co! Vdd! S!
MM1 Vdd! A _node_1 Vdd! PMOS w=5.4u l=1.2u
MM2 Vdd! B _node_1 Vdd! PMOS w=5.4u l=1.2u
MM3 Vdd! Ci _node_1 Vdd! PMOS w=5.4u l=1.2u
MM4 _node_1 Co! S! Vdd! PMOS w=5.4u l=1.2u
MM5 Vdd! A _node_2 Vdd! PMOS w=5.4u l=1.2u
MM6 _node_2 B _node_3 Vdd! PMOS w=5.4u l=1.2u
MM7 _node_3 Ci S! Vdd! PMOS w=5.4u l=1.2u
MM8 S! Co! _node_4 0 NMOS w=1.8u l=1.2u
MM9 _node_4 A 0 0 NMOS w=1.8u l=1.2u
MM10 _node_4 B 0 0 NMOS w=1.8u l=1.2u
MM11 _node_4 Ci 0 0 NMOS w=1.8u l=1.2u
MM12 S! Ci _node_5 0 NMOS w=1.8u l=1.2u
MM13 _node_5 A _node_6 0 NMOS w=1.8u l=1.2u
MM14 _node_6 B 0 0 NMOS w=1.8u l=1.2u
.ENDS
*** End Sub-Circuit

*** Sub-Circuit carry!
.SUBCKT   carry! A B Ci Vdd! Co!
MM1 Vdd! A _node_1 Vdd! PMOS w=5.4u l=1.2u
MM2 Vdd! B _node_1 Vdd! PMOS w=5.4u l=1.2u
MM3 Vdd! B _node_2 Vdd! PMOS w=5.4u l=1.2u
MM4 _node_1 Ci Co! Vdd! PMOS w=5.4u l=1.2u
MM5 _node_2 A Co! Vdd! PMOS w=5.4u l=1.2u
MM6 Co! Ci _node_3 0 NMOS w=1.8u l=1.2u
MM7 _node_3 A 0 0 NMOS w=1.8u l=1.2u
MM8 _node_3 B 0 0 NMOS w=1.8u l=1.2u
```

```
MM9 Co! A _node_4 0 NMOS w=1.8u l=1.2u
MM10 _node_4 B 0 0 NMOS w=1.8u l=1.2u
.ENDS
*** End Sub-Circuit

*** Sub-Circuit inverter
.SUBCKT   inverter A B Vdd!
MM1 Vdd! A B Vdd! PMOS w=5.4u l=1.2u
MM2 B A 0 0 NMOS w=1.8u l=1.2u
.ENDS
*** End Sub-Circuit
.end
* -- Extraction Completed --*
```

5. SUMMARY

The software tool desribed in this thesis, WebTop, is part of a larger framework to enable the Internet-based design, simulation, and fabrication of integrated circuits and systems. Specifically, this thesis involved the construction of a schematic editor and netlist extractor using Sun Microsystems' Java programming language. WebTop allows the user to draw a circuit schematic (cell) in a fully graphical interface and then to extract a netlist file suitable for simulation with various industry-standard circuit simulators, such as SPICE or Verilog. WebTop supports hierarchical structures: any given cell may be composed of other, lower-level cells. At the lowest level of abstraction, a large library of commonly used components is built in to WebTop to facilitate the creation of more complex cells. In addition to the built-in library of standard electronic components, additional component types may be created by the user and accessed either from local disk or a distributed network of World Wide Web (WWW) based object repositories (WebTop servers). Because WebTop is written entirely in Java, it should run on any platform that supports a Java virtual machine.

This is not to say that WebTop is a perfect application. It is new software, and use of any new software package will reveal features that are missing, features that are included but unnecessary, and other problems with the current version. In particular, the user interface is somewhat awkward in this version. Following is a brief list of possible future enhancements for WebTop.

First, WebTop should be updated to support the JDK1.1 specification. Although such an update is not practical at the time of this writing due to the lack of JDK1.1-compliant WWW browsers, the update will become necessary as JDK1.1 becomes standard. JDK1.1 promises to be more robust than JDK1.02, and implements many exciting new features such as printing, enhanced user interface components and keyboard accelerators, while also improving performance.

Other work should focus on extending WebTop's capabilities. In this version of WebTop, PrimitiveCells are fixed at compile time. It would be useful to have PrimitiveCells handled more abstractly, so that PrimitiveCells could be added, removed, or modified by the user, without having to recompile WebTop.

In this version of WebTop, SchematicCells always appear as a simple box. However, schematics could be made more clear if other shapes were allowed. Standard shapes for common circuits such as multiplexers, adders and multipliers exist. If such shapes could be associated with SchematicCells, the clarity of the schematics produced by WebTop would be greatly enhanced.

WebTop should be reorganized so that different netlist formats can be added easily, possibly even at runtime. Currently, implementing a new netlist format would involve adding a large amount of code, spread through many different class files.

APPENDIX A – WEBTOP USER MANUAL

WebTop is a schematic editor with netlist extraction capabilities. Basically, WebTop allows you to draw your designs on the screen, then have the design simulated, without having to worry about constructing the netlist yourself. WebTop has hierarchical features, so that your designs can incorporated into larger systems.

This document serves more as a reference guide than a tutorial. All the features and functions are listed here, so that you can look something up if necessary. In general, the best way to learn how to work with WebTop is to play with it.

Basics

First, we must introduce the various components of the user interface. The names indicated on the screen dumps here are used throughout the remainder of this document to identify user interface components or general regions of the screen.

Main Applet

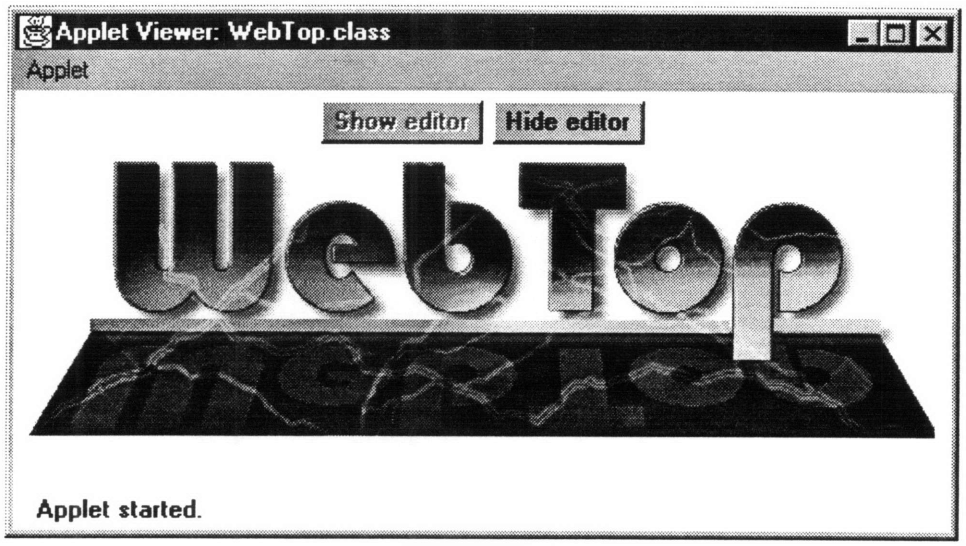
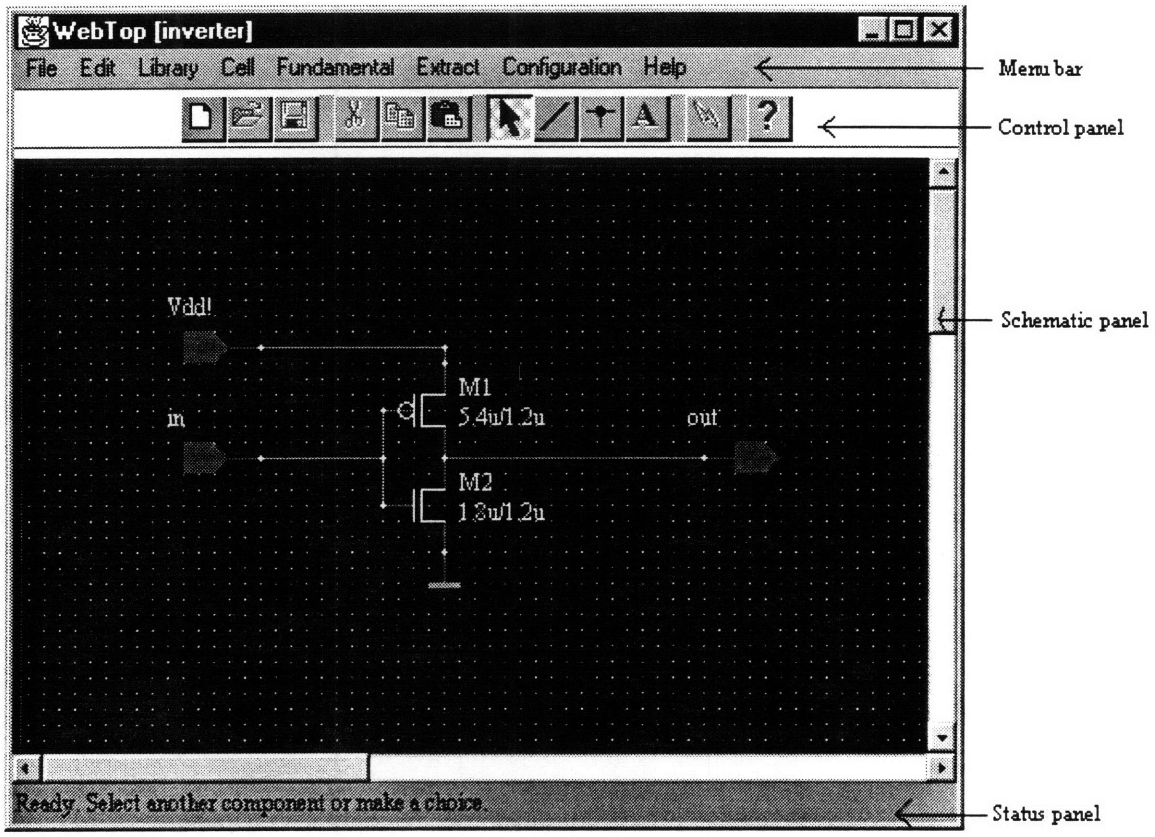


Figure 15: Main applet window

This is the main applet window. This (or something similar to it, depending on how exactly WebTop is invoked) is what appears when WebTop is accessed through a Web browser or appletviewer. The main applet window is not terribly useful; it merely provides an easy way to show or hide the schematic editor window.

Schematic Editor



Ready. Select another component or make a choice.

Figure 16: Schematic editor window

This is the schematic editor window. It contains several sub-components (labeled above) that will be frequently referred to throughout the remainder of this document.

Control panel

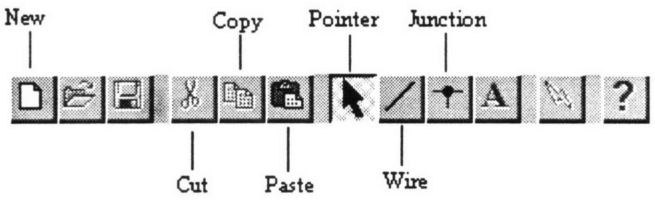


Figure 17: Control panel

The control panel provides a convenient method for performing certain operations. Any of the functions may be activated by clicking on the appropriate button.

If you forget which buttons perform which functions, a help message appears at the bottom of the screen when you position the mouse pointer over one of the buttons.

The New button starts a new cell. If the current cell has been modified since the last save, you will be prompted for confirmation before continuing. If you cancel, you are returned to the original cell. If you continue, or if the original cell had not been modified since the last save, you will be prompted for a new name. All components from the previous cell will be removed and the pin grid will be returned to its default 100x100 size. Clicking the new button is equivalent to selecting the New command from the File menu.

The Cut button will remove the selected components from the screen and place them into a buffer. This buffer can be retrieved through use of the Paste command. Whatever is in the buffer when the Cut command is run is overwritten, and cannot be recovered. The Cut button is enabled only when one or more components in the current cell are selected. If no components in the current cell are selected, the Cut button is grayed out and cannot be clicked. Clicking the Cut button is equivalent to selecting the Cut command from the Edit menu. It is also equivalent to pressing ‘d’ or ‘D’.

The Copy button will copy the selected components from the screen and place them into a buffer. This buffer can be retrieved through use of the Paste command. Whatever is in the buffer when the Copy command is run is overwritten, and cannot be recovered. The Copy button is enabled only when one or more components in the current cell are selected. If no components in the current cell are selected, the Copy button is grayed out and cannot be clicked. Clicking the Copy button is equivalent to selecting the Copy command from the Edit menu.

The Paste button will add any components in the buffer to the current cell. The components from the buffer will be placed at the location of the last mouse click on the SchematicPanel, if possible. If the paste location is too close to an edge, some of the newly added components will be relocated to keep them within the cell boundaries. Whatever is in the buffer when the Paste command is executed remains in the buffer. Repeated executions of the paste command will allow you to create repetitive structures easily. If your cell contains many repeated structures, using hierarchy is probably a better idea. Clicking the Paste button is equivalent to selecting the Paste command from the Edit menu.

The Pointer button puts the editor in pointer mode. In pointer mode, you can select components, move components, and change component's properties. To select a component, position the pointer over the main body of the component and click. To select multiple components, hold shift down while clicking on each component. To select all the components in a particular region, click the mouse on one corner of the region to select, and drag the mouse to the opposite corner of the region. All components contained within the box will be selected. To move a component, click on the main body of the component, and drag the mouse to the new location. If you select multiple components, you can move them as a group in the same manner as a single component. To change a component's properties, double click on the main body of the component to pop up the component's PropertyBox. Clicking the Pointer button is equivalent to selecting the Pointer Mode command from the Fundamental menu. It is also equivalent to clicking the mouse on the status panel at the bottom of the screen.

The Wire button puts the editor in wire drawing mode. Wire drawing mode is specifically for adding new wires to the cell. To add a new wire, click the mouse on one endpoint, and drag to the other endpoint. If wires are drawn crossing, they will not connect by default. If a wire crosses across the end of a component's pin, it will not connect by default. If an endpoint of the wire is placed on a component's pin, or on another wire, then a connection will take place. Connections are indicated by round junctions. If no junction is present, no connection is made. Clicking the Wire button is equivalent to selecting the Wire Mode command from the Fundamental menu.

The Junction button puts the editor in junction drawing mode. Junction drawing mode is specifically for adding junctions to the cell. Junctions can only be added at grid points on the interior of an existing wire. The restriction that junctions may only be placed on grid points makes it difficult, if not impossible, to place junctions on a wire that is not vertical or horizontal. When a junction is added, the involved wire is broken into two wires, each of which can then be selected and moved independently. Clicking the Junction button is equivalent to selecting the Junction Mode command from the Fundamental menu.

Schematic editor menus

Most of the features of WebTop are accessible from the schematic editor's menus.

File menu

The New command starts a new cell. If the current cell has been modified since the last save, you will be prompted for confirmation before continuing. If you cancel, you are returned to the original cell. If you continue, or if the original cell had not been modified since the last save, you will be prompted for a new name. All components from the previous cell will be removed and the pin grid will be returned to its default 100x100 size. Selecting the New command from the File menu is equivalent to clicking the New button in the control panel.

The PostScript command produces a PostScript output file of the current cell. This functionality is available only if the applet has permission to write to the local disk. Sun's appletviewer can be configured to allow local disk access; Netscape Navigator 3.x and Microsoft Internet Explorer 3.x cannot. The PostScript output will be automatically scaled to fit on one page with half-inch margins. WebTop will automatically choose landscape or portrait mode based on orienting the longer edge of the cell to the longer edge of the page. For square cells, the default orientation is portrait.

The Close command hides the entire editor window. It does not stop the applet from running, and no data is lost. The editor window can be re-displayed by pressing the "Show editor" button in the main applet window. The Close command is equivalent to pressing the "Hide editor" button in the main applet window.

Edit menu

The Cut command will remove the selected components from the screen and place them into a buffer. This buffer can be retrieved through use of the Paste command. Whatever is in the buffer when the Cut command is run is overwritten, and cannot be recovered. The Cut command is enabled only when one or more components in the current cell are selected. If no components in the current cell are selected, the Cut command is grayed out and cannot be selected. Selecting the Cut command is equivalent to clicking the Cut button in the control panel. It is also equivalent to pressing 'd' or 'D'.

The Copy command will copy the selected components from the screen and place them into a buffer. This buffer can be retrieved through use of the Paste command. Whatever is in the buffer when the Copy command is run is overwritten, and cannot be recovered. The Copy command is enabled only when one or more components in the current cell are selected. If no components in the current cell are selected, the Copy command is grayed out and cannot be selected. Selecting the Copy command is equivalent to clicking the Copy button in the control panel.

The Paste command will add any components in the buffer to the current cell. The components from the buffer will be placed at the location of the last mouse click on the SchematicPanel, if possible. If the paste location is too close to an edge, some of the newly added components will be relocated to keep them within the cell boundaries. Whatever is in the buffer when the Paste command is executed remains in the buffer. Repeated executions of the paste command will allow you to create repetitive structures easily. If your cell contains many repeated structures, using hierarchy is probably a better idea. Selecting the Paste command is equivalent to Clicking the Paste button in the control panel.

The Select All command selects all of the components in the current cell. This is equivalent to manually selecting each component, or drawing the select box around the entire pin grid area.

The Zoom In command causes everything to look larger. This is for display purposes only; actual dimensions of the pin grid or of the components themselves are not affected. There is no limit (subject to system memory constraints) on how far you may zoom in. The Zoom In command is equivalent to pressing 'z'.

The Zoom Out command causes everything to look smaller. This is for display purposes only; actual dimensions of the pin grid or of the components themselves are not affected. Zoom out is useful for working with large cells. The minimum size is achievable is four pixels per grid cell; further Zoom Out commands will have no further effect. The Zoom Out command is equivalent to pressing 'Z'.

The Expand Horizontal command increases the size of the pin grid by fifty pins in the horizontal direction. The new pins are added at the right edge of the current cell. The positions of components already in the current cell are not affected. There is no limit (subject to system memory constraints) on how large the pin grid may become in the horizontal direction. However, it is important to note that WebTop will slow down as the size of the pin grid is increased. It is best to increase the size of the pin grid only as needed. If the cell you are working on becomes very large, use of hierarchy is usually a good way to decrease the area required for the cell.

The Expand Vertical command increases the size of the pin grid by fifty pins in the vertical direction. The new pins are added at the bottom edge of the current cell. The positions of components already in the current cell are not affected. There is no limit (subject to system memory constraints) on how large the pin grid may become in the vertical direction. However, it is important to note that WebTop will slow down as the size of the pin grid is increased. It is best to increase the size of the pin grid only as needed. If the cell you are working on becomes very large, use of hierarchy is usually a good way to decrease the area required for the cell.

Library menu

The Show LibMgr command displays the library manager on the screen. The library manager is discussed in a separate section of this document.

The Hide LibMgr command removes the library manager from the screen. The library manager is discussed in a separate section of this document.

Cell menu

The Save Cell command saves the current cell in the library manager. This does not mean that the cell has been saved permanently! This means only that the cell is now available in the library manager. Without further action (in the library manager) the cell will be lost when WebTop is closed. If the cell name of the current cell is not already present in the library, it is added. If the cell name of the current cell is already present, the cell already in the library is overwritten and cannot be recovered.

The Change Name command changes the name of the current cell. Any component names left at the default value (CELLNAME_COMPONENTNAME_ID#) are updated to reflect the name change. This command only affects the working copy of the current cell; no changes are made to cells stored by the library manager.

The Spice View command allows the user to alter the cell's spice view. This command will pop up a window with a text area for editing the spice view. If no spice view is defined for the current cell, a dialog will appear to note that fact. Buttons to add the view, delete the view, and cancel are provided. The Add View button will set the view of the current cell to the text present in the text area. The Delete View button will remove the spice view for the current cell. The Cancel button closes the view window and leaves the spice view of the current cell unchanged.

The Verilog View command allows the user to alter the cell's Verilog view. This command will pop up a window with a text area for editing the Verilog view. If no Verilog view is defined for the current cell, a dialog will appear to note that fact. Buttons to add the view, delete the view, and cancel are provided. The Add View button will set the view of the current cell to the text present in the text area. The Delete View button will remove the Verilog view for the current cell. The Cancel button closes the view window and leaves the Verilog view of the current cell unchanged.

Fundamental menu

Unless otherwise noted, all of the commands in the Fundamental menu add the correspondingly named PrimitiveCell to the current cell. For example, the R command adds a resistor, the L command adds an inductor, and so on.

The Pointer Mode command puts the editor in pointer mode. In pointer mode, you can select components, move components, and change component's properties. To select a component, position the pointer over the main body of the component and click. To select multiple components, hold shift down while clicking on each component. To select all the components in a particular region, click the mouse on one corner of the region to select, and drag the mouse to the opposite corner of the region. All components contained within the box will be selected. To move a component, click on the main body of the component, and drag the mouse to the new location. If you select multiple components, you can move them as a group in the same manner as a single component. To change a component's properties, double click on the main body of the component to pop up the component's PropertyBox. Selecting the Pointer Mode command is equivalent to clicking the Pointer button in the control panel. It is also equivalent to clicking the mouse on the status panel at the bottom of the screen.

The Wire Mode command puts the editor in wire drawing mode. Wire drawing mode is specifically for adding new wires to the cell. To add a new wire, click the mouse on one endpoint, and drag to the other endpoint. If wires are drawn crossing, they will not connect by default. If a wire crosses across the end of a component's pin, it will not connect by default. If an endpoint of the wire is placed on a component's pin, or on another wire, then a connection will take place. Connections are indicated by round junctions. If no junction is present, no connection is made. Selecting the Wire Mode command is equivalent to clicking the Wire button in the control panel.

The Junction Mode command puts the editor in junction drawing mode. Junction drawing mode is specifically for adding junctions to the cell. Junctions can only be added at grid points on the interior of an existing wire. The restriction that junctions may only be placed on grid points makes it difficult, if not impossible, to place junctions on a wire that is not vertical or horizontal. When a junction is added, the involved wire is broken into two wires, each of which can then be selected and moved independently. Selecting the Junction command is equivalent to clicking the Junction button in the control panel.

The PrimitiveCells in the External Pin menu are placed like any other of the PrimitiveCells available from the Fundamental menu. However, these particular cells are special. These cells (isoInputPin, isoOutputPin, and isoInputOutputPin) specify an interface to the external world. In order to use the cell you are designing as a part of larger cells, you must specify the inputs and outputs accessible from the rest of the world. Each of these three cells implements a different type of pin. The functions are the same;

the type controls only how the pins are drawn. The isoInputPin is an input from the external world to the cell. The isoOutputPin is an output from the cell to the external world. The isoInputOutputPin is used for bi-directional signals. IsoInputPins become the input pins of the SchematicCell and are drawn on the left side of the SchematicCell, with arrows pointing in towards the main body of the SchematicCell. IsoOutputPins become the output pins of the SchematicCell and are drawn on the right side of the SchematicCell, with arrows pointing away from the main body of the SchematicCell. IsoInputOutputPins become the input/output pins of the Schematic and are drawn on the right side of the SchematicCell, with no arrows. Note that each of these special IOPin structures must be connected to a different circuit node. Attempts to connect the same node to multiple External Pin structures will result in an error.

Extract menu

The Spice Models sub-menu contains a special kind of menu item. Instead of the usual menu item, which executes a command when selected, these menu items change state. If the menu item appears with a check, it is "on" or "true." If no check appears, the menu item is "off" or "false." Each of these special menu items corresponds to a commonly used, predefined set of spice .MODEL statements. If the menu item is in the "on" state at extraction time, the corresponding spice .MODEL statements are included in the output file. Only one set of .MODEL statements may be included; Selection of one set will automatically disable all other sets. By default, no .MODEL statements are included.

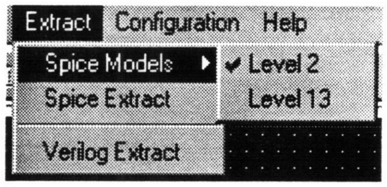


Figure 18: Spice Models menu items

The Spice Extract command creates a spice netlist from the current cell and displays the netlist on the screen. If errors are found while extracting the spice netlist, they are noted in the spice output, and presented via dialog box.

The Verilog Extract command creates a Verilog netlist from the current cell and displays the netlist on the screen. If errors are found while extracting the Verilog netlist, they are noted in the Verilog output, and presented via dialog box.

Configuration menu

The Show Color Manager command displays the color manager on the screen. The color manager is discussed in a separate section of this document.

The Hide Color Manager command removes the color manager from the screen. The color manager is discussed in a separate section of this document.

Help menu

The options in the Help menu are currently disabled. They are reserved for future versions of WebTop.

Library Manager

The library manager is responsible for collecting and organizing libraries of cells. The library manager provides functionality for loading and saving cells, both to local disk and to Internet-based servers.

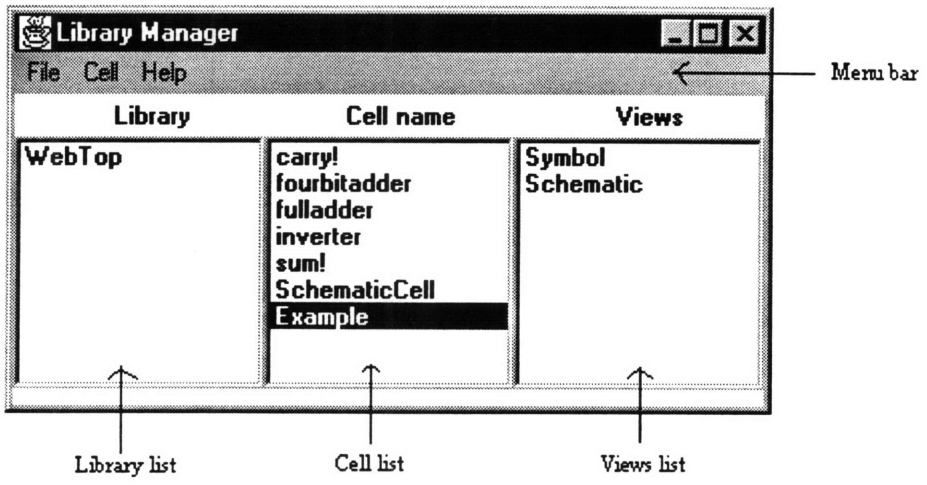


Figure 19: Library manager

Mouse-Based functions

Some of the functions of the library manager are available only through mouse control. Other options are available both by mouse control and menu commands. In terms of mouse control, functions are activated by either clicking or double clicking on elements of the three lists contained by the library manager.

Currently, WebTop supports only one library. Thus, clicking or double clicking in the library list has no effect.

Clicking on a cell name in the cell list will select that cell. While not a useful thing to do by itself, several of the menu commands act upon whatever cell is selected in the cell list. Double clicking on a cell name in the cell list is equivalent to selecting the Add Cell command from the Cell menu.

Clicking on a view name in the views list will select that view. Double clicking on a view name will actually take action. Double clicking on the Symbol view in the views list will display a representation of the symbol view of the selected cell.. Figure 20 below shows an example of a symbol view.

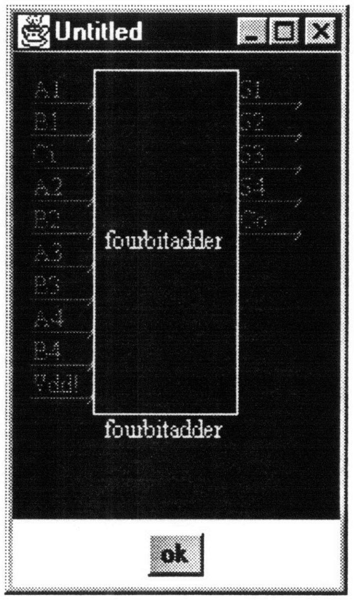


Figure 20: Example of a symbol view

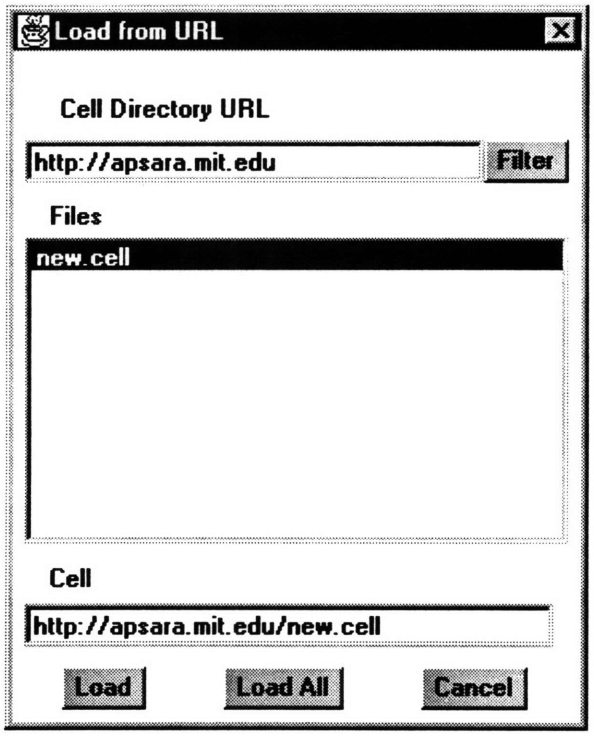
Double clicking on the Schematic view in the views list is equivalent to selecting the Edit Cell command from the Cell menu. If a SPICE or Verilog view is present for the selected cell, the view may be viewed, but not edited, by double clicking the appropriate view name in the view list.

Library manager menus

Most of the functions of the library manager are accessible from the menus.

File menu

The Load Cell sub-menu contains a variety of commands for loading cells. The Load From File command loads a WebTop-format cell file from the local disk (if the Java environment WebTop is running in allows local disk access). The file to load is chosen using a file dialog box specific to the operating system that you are running WebTop under. The Load as Object command loads an RMI-format cell file from the local disk. As with the Load Cell command, a local OS-specific file dialog box is used for file selection. The Load All command will attempt to load all of the files in a specified directory as WebTop-format cells. If files not conforming to the WebTop format are present in the specified directory, errors will result. In the JDK1.02 specification, there is no way to select a directory using the local OS-specific dialog boxes. Thus, to use the Load All command, you must select a particular file in the target directory. JDK1.02 does not allow the selection of directories, only of files. Even though a specific file in the directory must be selected, all files in the same directory as the selected file will be loaded. The Load URL command loads a cell from a specified URL, using a special URL dialog box, show in Figure 21.


Load from URL

Cell Directory URL

http://apsara.mit.edu

Files

new.cell

Cell

http://apsara.mit.edu/new.cell

Figure 21: URL dialog box

The URL dialog box allows specification of a WebTop server, and selection of the cell or cells to load.

The Save to File command will save a WebTop-format representation of the specified cell to local disk (if the Java environment WebTop is running in allows local disk access). The cell to be saved is selected by clicking on the cell name in the library manager's cell list. WebTop will provide a local OS-specific file dialog to choose a location and filename.

The Save to Object command is equivalent to the Save to File command, except that the Save to Object command will save an RMI-format representation of the cell.

The Save All command is similar to the Save to File command. However, instead of saving only the selected cell, all cells in the current library will be saved.

RMI-format cells load and save more quickly than the WebTop representations, but take approximately three times as much space. Furthermore, RMI-format cells may not be compatible across different platforms, and may not be compatible between different WebTop versions. Thus, RMI-format is best used while developing cells, but WebTop format should be used for long term compatibility purposes.

Cell menu

The New Cell command starts a new cell. If the current cell has been modified since the last save, you will be prompted for confirmation before continuing. If you cancel, you are returned to the original cell. If you continue, or if the original cell had not been modified since the last save, you will be prompted for a new name. All components from the previous cell will be removed and the pin grid will be returned to its default 100x100 size. Selecting the New Cell command is equivalent to selecting the New command from the File menu of the schematic editor window, or to pressing the New button in the control panel of the schematic editor window.

The Add Cell command adds a SchematicCell from the library manager to the current cell. The currently selected cell in the library manager's cell list is added to the current cell as a symbol view. That is, the cell is added as a "black box" with the proper inputs and outputs. The contents of a cell added to a cell this way are not accessible to the user. This command is analogous to adding a PrimitiveCell from the schematic editor window. Figure 22 below shows the result of using Add Cell command to add an inverter cell to a new cell called "Example".

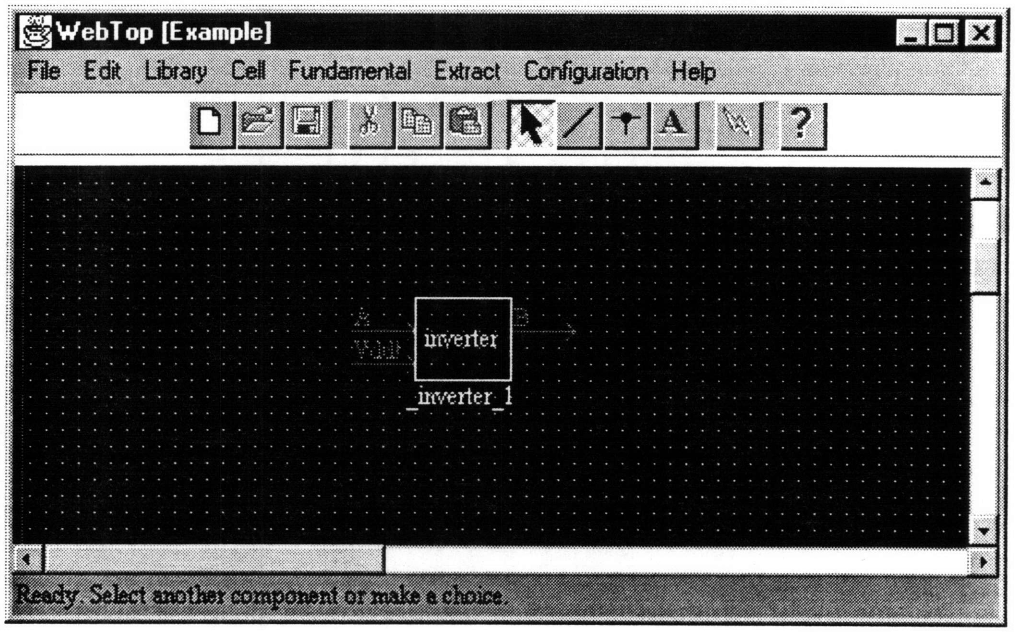
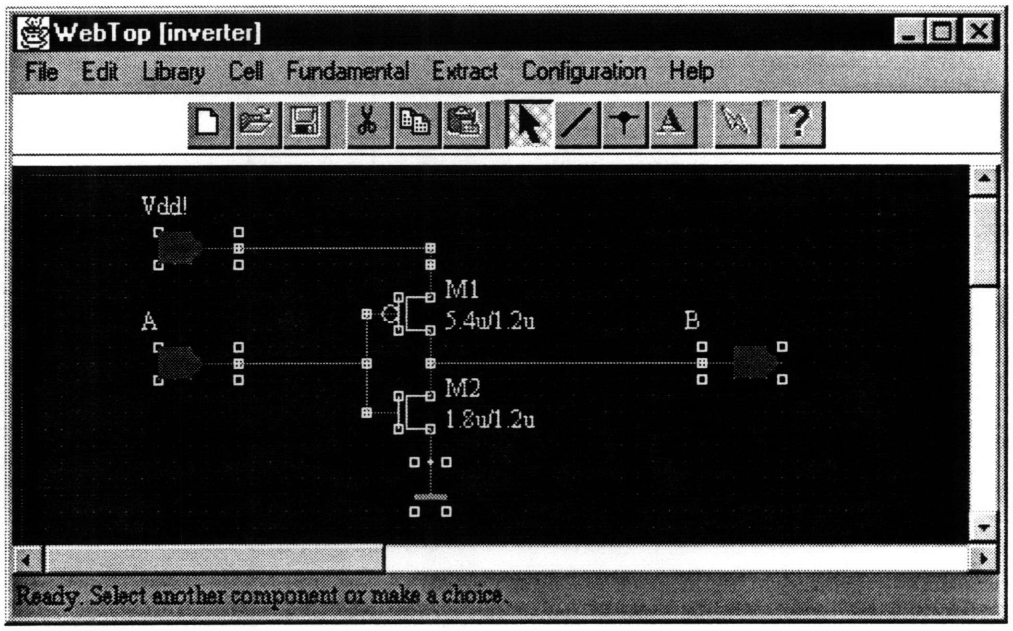


Figure 22: Example of the Add Cell command

The Edit Cell command allows the user to alter the internals of a SchematicCell. The currently selected cell in the cell list is the target of this command. If the current cell has been modified since the last save, you will be prompted for confirmation before continuing. If you cancel, you are returned to the original cell. If you continue, or if the current cell has not been modified since the last save, the current cell will be discarded. The internal working of the selected SchematicCell are displayed in full detail, and can be modified in any way. Note that modifications to the cell do not take effect in the library manager unless the cell is explicitly saved. Figure 23 below shows the result of editing the same inverter cell that was added using "Add Cell" before.


WebTop [inverter]
File Edit Library Cell Fundamental Extract Configuration Help
Vdd!
A
M1
5.4u/1.2u
B
M2
1.8u/1.2u

Ready. Select another component or make a choice.

Figure 23: Example of the Edit Cell command

The Delete Cell command removes a cell from the library. The currently selected cell in the cell list is the target of this command. Note that this command simply removes the selected cell from the library manager; it does not affect any copies of the cell saved on disk or on remote WebTop servers.

Help menu

The Help menu has no options at this time; however, it will be expanded in future versions of WebTop.

Color Manager

The color manager allows the user to configure the colors used to display the various structures of the WebTop interface.

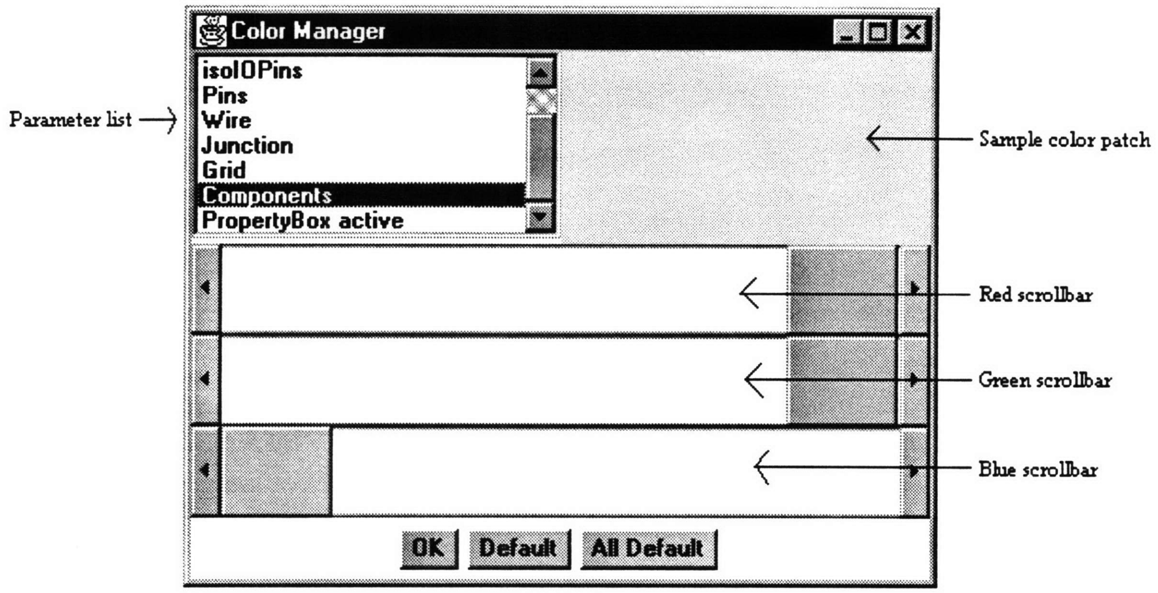


Figure 24: Color manager window

The color manager contains three major components: the scrollbars, the sample color patch, and the parameter list. The parameter list contains the names of every component with a user-configurable color. The sample color patch display the color of the currently selected item in the parameter list. The scrollbars are used to set the color. There are three scrollbars, each for one component of the color. The top scrollbar is for red, the middle for green, and the bottom for blue. Figure 24 shows that red plus green equals yellow in the RGB color space. To set a color, click on the appropriate target parameter in the parameter list, and slide the scroll bars until the desired color is achieved. For most parameters, the color in the schematic editor will change as the scrollbars are manipulated; however, for efficiency reasons, the Background and Grid colors will be updated onscreen only when the color manager is closed. To return any component to its original default color, select the appropriate parameter from the parameter list and click the Default button. To return all parameters to their default values, click the All Default button.

Property box

The property box is used to modify the parameters of a cell. The property box is displayed when a component is double clicked, or when the component is initially placed. Every cell has a least two parameters, name and instanceName. The name parameter is simply the name of the component associated with the property box, and may be changed. The instanceName reflects the type of Cell that this property box is associated with, and may not be changed. Cells may optionally have other parameters, such as value, length, width, and so on. These optional parameters control how the component is represented when extracted to a Verilog or SPICE format netlist.

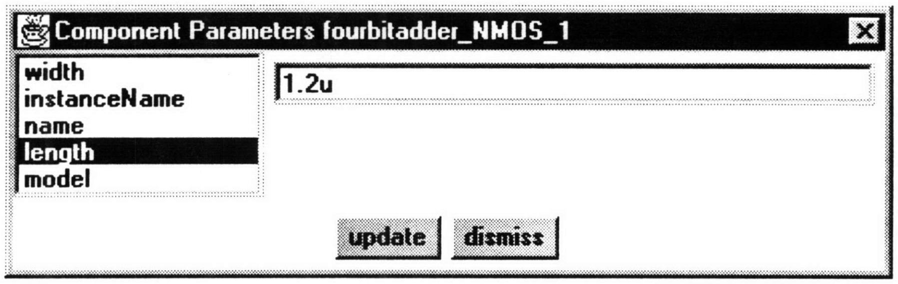


Figure 25: Property box

To view or alter a particular parameter, click on its name in the list at left. The current value of the selected parameter will appear in the text field. To change the value of the selected parameter, type the new value in the text field, and press return. The appropriate component value will be updated, and the property box will remain visible. Another way to update the parameter is to type a new value and click the update button. The appropriate parameter will be updated, and the property box will be dismissed. In order to dismiss the property box without updating anything, click the dismiss button. There is no need to dismiss the PropertyBox when finished. When the PropertyBox is given the input focus, the SchematicPanel's viewing area is adjusted so that the associated component is visible onscreen, and highlighted with PropBoxActive-style highlighting.

APPENDIX B— WEBTOP FILE FORMAT

As discussed elsewhere in this document, WebTop supports two different types of cells, PrimitiveCells and SchematicCells. PrimitiveCells implement the fundamental building blocks of a circuit, such as resistors, transistors, and so on. SchematicCells, however, are more complicated. A SchematicCell is a single unit, composed of other PrimitiveCells and SchematicCells. For example, a CMOS inverter would be a SchematicCell – it would contain an NMOS transistor, a PMOS transistor, voltage sources, and perhaps other components. Connecting several inverter cells together to make a ring oscillator would result in another SchematicCell.

Due to the fundamental nature of PrimitiveCells, they are implemented as Java classes. As of this writing, the PrimitiveCells available to the user is set at compile time. PrimitiveCells cannot be added or removed from this version of WebTop at runtime; any such changes require altering the source code and recompiling. Thus, there is no file format for PrimitiveCells.

SchematicCells, however, are defined by the user. SchematicCells can be loaded and saved to local disk, or loaded from a remote WebTop server. Thus, some file format is necessary for the storage of SchematicCells. Perhaps the easiest solution to this problem would be to use Java's RMI features to store the actual SchematicCell objects. In practice, this solution is less than optimal. First, there is the potential for objects to be incompatible across Java implementations. Although objects should nominally be portable across different Java implementations, serious cross-platform compatibility bugs were discovered during testing. A more serious problem with the use of RMI is future compatibility. Use of the RMI features in this way would result in output files that would work only with the particular version of WebTop that created them. If, in the next release of WebTop, the object definition of SchematicCell should change, all previously existing SchematicCell files would become incompatible with the new version. Thus, a special format has been developed to store SchematicCells in. This format has been design with future compatibility in mind. RMI storage options are also provided, for the RMI format is much faster to load and save. However, the data files created with the RMI format are approximately three times larger than the WebTop format for the same cell. The rest of this appendix is concerned with the WebTop file format.

A SchematicCell output file is plain ASCII text. Every line in the SchematicCell output file is terminated with an ASCII '/n' (newline) character. Lines are case-sensitive; "This is a line" is not considered equivalent to "this is a line." Lines can be grouped into three classes: version, separators, and parameters. Version lines indicate the version of the WebTop file format a given file is written for. Separators are used to arrange parameter lines into smaller, logical groupings.

Parameter lines contain a string or a number that represent some aspect of the SchematicCell. In the files, parameter lines are not labeled. The position of the parameter line relative to the appropriate separator lines determines what parameter the line represents.

Separator lines always come in pairs. The first line of the pair is of the form "@Begin SEPARATORNAME@", and the second is of the form "@End

SEPARATORNAME@" Between the opening half and closing half of a separator pair, zero or more parameter lines may be enclosed. Seven different types of separators are defined: TopCell, Basics, Spice View, Verilog View, Parameters, Pins, and SubCell.

TopCell separators surround the entire file. In the current version, they are ignored; They are included in case future versions allow different types of cells to be saved.

Basics separators enclose global parameters of the cell. Basics parameters are used to set the correspondingly name members of the SchematicCell class. These parameters are, in order : instanceName (String), totalGridX (int), totalGridY (int), spiceViewPresent ("true" | "false"), verilogViewPresent ("true" | "false").

Spice View separators enclose the spice view of the cell. These separators are required because the spice view is a string of arbitrary length. Anything between Spice View separator lines is assumed to be part of the cell's spice view.

Verilog View separators enclose the Verilog view of the cell. These separators are required because the Verilog view is a string of arbitrary length. Anything between Verilog View separator lines is assumed to be part of the cell's Verilog view.

Parameters separators enclose the parameters defined for a cell. More specifically, the contents of the cell's Parameters hashtable are enclosed here. The contents of the hashtable are put into the file in key, value pairs. Thus, the first line after the opening half of the Parameters separator is a key, the next line is the corresponding value, the third line is a new key, and so on. Parameters are read in this fashion until the block is terminated by the closing half of the Parameters separator.

Pins separators enclose a number of pins, and a list of pin descriptions. These pin descriptions correspond to the IOPins in a cell's pins Vector. The first line following the opening half of the Pins Separator line is an integer indicating the number of pin descriptions to follow. Next is the series of pin descriptions. The pin descriptions depend on what type of cell is being described. For a SchematicCell, each pin description contains: type (int), lexical_name (String), pin_name (String). For a PrimitiveCell, each pin description contains type (int) and lexical_name (String). The pin_name is not required for the PrimitiveCell because the pin_name is set by the definition of the PrimitiveCell, and never changes.

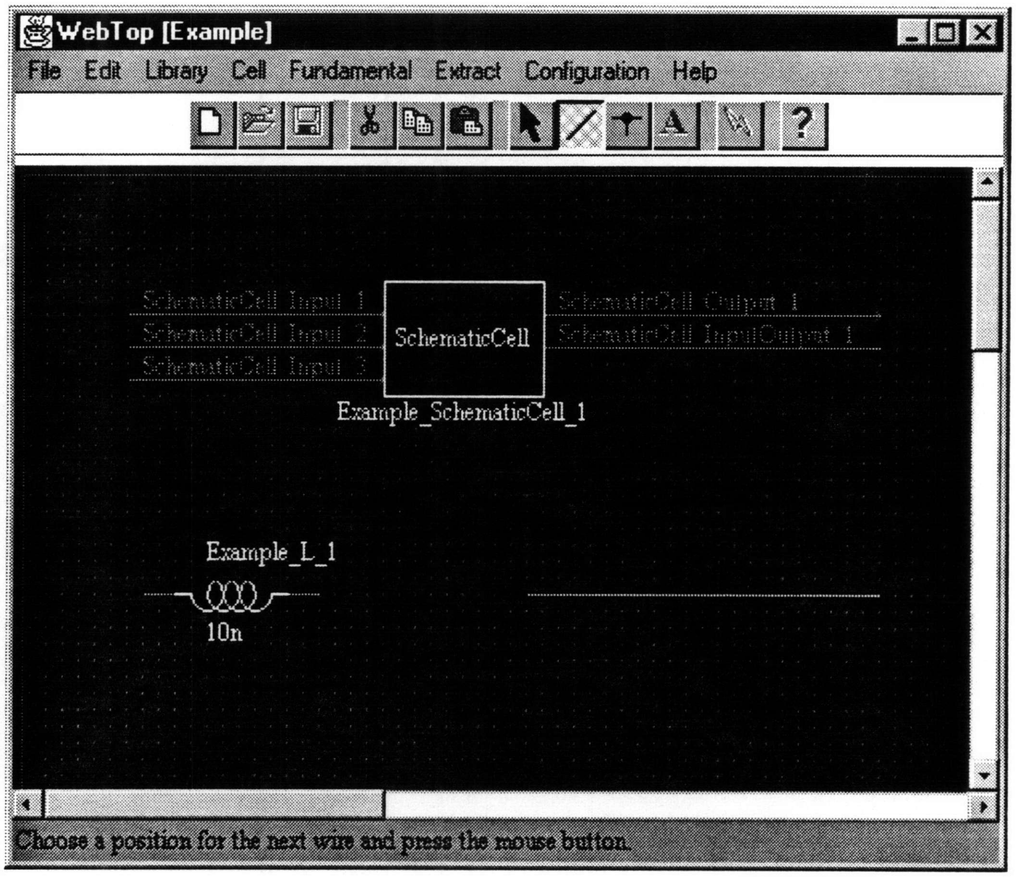
SubCell separators enclose descriptions of each subcell of a SchematicCell. In the actual SchematicCell object, these are the Cells that would be found in the subcells Vector. Because there may be different types of Cells in the subcells Vector (PrimitiveCell, SchematicCell, Wire), there are three different formats that may be enclosed within a SubCell separator.

A SubCell block for a PrimitiveCell contains: "PrimitiveCell" (literal String), instanceName (String), Pos.x (int), Pos.y (int), a Parameters separator block, and a Pins separator block.

A SubCell block for a SchematicCell contains: "SchematicCell" (literal String), instanceName (String), Pos.x (int), Pos.y (int), a Parameters separator block, and a Pins separator block.

A SubCell block for a Wire contains: x1 (int), y1 (int), x2 (int), y2 (int). These indicate the coordinates of each end of the Wire.

The example cell shown in Figure 26 will demonstrate the use of each type of separator and parameter line. It contains a SchematicCell, a PrimitiveCell (inductor) and a wire, for demonstration purposes.



Choose a position for the next wire and press the mouse button.

Figure 26: Example Cell for file format

When saved as a WebTop-format file, the cell looks like this:

```
@WebTop Version 1.01@
@Begin TopCell@
@Begin Basics@
Example
100
100
false
false
@End Basics@
@Begin Spice View@

@End Spice View@
@Begin Verilog View@
```

```
@End Verilog View@
@Begin SubCell@
SchematicCell
SchematicCell
6
6
@Begin Parameters@
instanceName
SchematicCell
name
Example_SchematicCell_1
@End Parameters@
@Begin Pins@
5
1
_node_1
_SchematicCell_Input_1
1
_node_2
_SchematicCell_Input_2
2
_node_3
_SchematicCell_Output_1
3
_node_4
_SchematicCell_InputOutput_1
1
_node_5
_SchematicCell_Input_3
@End Pins@
@End SubCell@
@Begin SubCell@
PrimitiveCell
L
7
24
@Begin Parameters@
instanceName
L
value
10n
name
Example_L_1
@End Parameters@
@Begin Pins@
2
3
_node_6
3
```

```
_node_7
@End Pins@
@End SubCell@
@Begin SubCell@
Wire
32
26
54
26
@End SubCell@
@End TopCell@
```

APPENDIX C – CLASS FILE REFERENCE GUIDE

This appendix contains an alphabetical listing of the class files that compose WebTop and a brief description of each class' purpose.

- AND.class – the AND class is a PrimitiveCell class. It represents a two-input and logic gate. This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.
- Buffer.class – the Buffer class is a PrimitiveCell class. It represents a single-input buffer logic gate. This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.
- C.class – the C class is a PrimitiveCell class. It represents a capacitor. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- Cell.class – the Cell class is a cell-related data structure class. The Cell class is the basic building block of anything designed in WebTop. Any component that can be added to the SchematicPanel is a Cell. Cells come in three varieties: PrimitiveCells, SchematicCells, and Wires.

- ColorManager.class – the ColorManager class is a user interface class. The ColorManager is used to specify the colors used to display various aspects of the WebTop interface.
- ComponentBox.class – the ComponentBox class is a user interface class. The ComponentBox is used by the LibMgr class to display the symbol view of a SchematicCell from the cell library.
- ComponentPanel.class – the ComponentPanel is a user interface class. The ComponentPanel is used by the LibMgr class to display the symbol view of a SchematicCell from the cell library.
- ControlPanel.class – the ControlPanel is a user interface class. The ControlPanel is a collection of mouse-clickable buttons that provide shortcuts to common commands such as cut, copy, paste, etc.
- DialogPanel.class – The DialogPanel is a user interface class. The DialogPanel contains the text, buttons, and optional graphic image of the SimpleDialog class.
- E.class – the E class is a PrimitiveCell class. It represents a voltage-controlled voltage source. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- ExtractDialog.class – the ExtractDialog class is a user interface class. The ExtractDialog is used to display the SPICE or Verilog netlist when it is extracted by WebTop. The ExtractDialog also allows submission of the netlist to other Web-based tools.

- G.class – the G class is a PrimitiveCell class. It represents a voltage-controlled current source. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- GND.class – the GND class is a PrimitiveCell class. It represents a connection to ground. Any circuit node connected to a GND cell will be named '0'. The GND cell should not appear in either SPICE or Verilog netlists; the netlist extractor should automatically ignore GND cells.
- HelpDialog.class – the HelpDialog class is a user interface class. It is used to display an online help message to the user. In this version of WebTop, online help is disabled. The HelpDialog class has been kept because the online help functionality should be added in later versions of WebTop, as new help files are written.
- I.class – the I class is a PrimitiveCell class. It represents an independent current source. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- ImageButton.class – the ImageButton class is a user interface class. The ControlPanel is a collection of ImageButtons; each individual button in the ControlPanel is an instance of the ImageButton class.
- imagecanvas.class – the imagecanvas class is a user interface class. The imagecanvas class is used to display the fancy WebTop graphic in the main applet window.

- InputBox.class – the InputBox class is a user interface class. The InputBox is used to prompt the user for a new cell name. The InputBox will block input to all other user interface features while it is active (it is a modal dialog box).
- Inverter.class – the Inverter class is a PrimitiveCell class. It represents a single-input inverter logic gate (not gate). This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.
- IOPin.class – the IOPin class is a data structure class. It represents a connection point to a Cell object.
- isoInputOutputPin.class – the isoInputOutputPin class is a PrimitiveCell class. It is used in the schematic view of a SchematicCell to specify a bi-directional input/output pin in the symbol view. The isoInputOutputPin cell should not appear in either SPICE or Verilog netlists; the netlist extractor should automatically ignore isoInputOutputPin cells.
- isoInputPin.class – the isoInputPin class is a PrimitiveCell class. It is used in the schematic view of a SchematicCell to specify an input pin in the symbol view. The isoInputPin cell should not appear in either SPICE or Verilog netlists; the netlist extractor should automatically ignore isoInputPin cells.
- isoOutputPin.class – the isoOutputPin class is a PrimitiveCell class. It is used in the schematic view of a SchematicCell to specify an output pin in the symbol view. The isoOutputPin cell should not appear in either SPICE or Verilog netlists; the netlist extractor should automatically ignore isoOutputPin cells.

- L.class – the L class is a PrimitiveCell class. It represents an inductor. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- LibCell.class – the LibCell class is a data structure class. It is used by the LibMgr class to keep track of the Cell objects loaded in the current cell library.
- LibMgr.class – the LibMgr class is a user interface class. It keeps track of cells in the library, loads and saves cells to local disk and network, and performs other functions related to cell management.
- NAND.class – the NAND class is a PrimitiveCell class. It represents a two-input nand logic gate. This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.
- NMOS.class – the NMOS class is a PrimitiveCell class. It represents a three-terminal n-channel MOSFET. In a SPICE netlist, the bulk terminal of the MOSFET is automatically connected to the "0" node (global ground). This PrimitiveCell class supports both SPICE and Verilog extraction.
- NMOS4.class – the NMOS4 class is a PrimitiveCell class. It represents a four-terminal n-channel MOSFET. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- NOR.class – the NOR class is a PrimitiveCell class. It represents a two-input nor logic gate. This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.

- OR.clas – the OR class is a PrimitiveCell class. It represents a two-input or logic gate. This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.
- Pin.class – the Pin class is a data structure class. It represents a point in space for components to connect to. Each grid point drawn in the SchematicPanel roughly corresponds to a Pin object.
- PMOS.class – the PMOS class is a PrimitiveCell class. It represents a three-terminal p-channel MOSFET. In a SPICE netlist, the bulk terminal of the MOSFET is automatically connected to the "Vdd!" node. It is the user's responsibility to define the Vdd! Node. This PrimitiveCell class supports both SPICE and Verilog extraction.
- PMOS4.class – the PMOS4 class is a PrimitiveCell class. It represents a four-terminal p-channel MOSFET. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- PrimitiveCell.class – the PrimitiveCell class is a cell-related data structure class. The PrimitiveCell class defines the type of cell that comes included with WebTop.
- PropertyBox.class – the PropertyBox is a user interface class. It is used to set the parameters of a cell, such as name, value, and so on.
- PropertyField.class – the PropertyField is a user interface class. It is used by the PropertyBox class only.

- PSGr.class – the PSGr class is a user interface class. It is an extension of the Graphics class that images to Adobe PostScript code instead of to the screen. It is used to generate PostScript files.
- R.class – the R class is a PrimitiveCell class. It represents a resistor. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- repaintcanvas.class – the repaintcanvas class is a user interface class. It is used by the ColorManager to display a color sample.
- Schematic.class – the Schematic class is a cell-related data structure class. It represents an arbitrary collection of cells. The Schematic class is used to track all of the cells present, all of the selected cells present, and in cut/copy/paste operations.
- SchematicCell.class – the SchematicCell class is a cell-related data structure class. A SchematicCell is one that can contain other cells. The SchematicCell is usually encountered in its "black box" symbol view form.
- SchematicPanel.class – the SchematicPanel class is a user interface class. It is the main work area of WebTop. It contains the schematic (cell) currently being edited.
- SimpleDialog.class – the SimpleDialog class is a user interface class. It is used to display errors and messages to the user in a pop-up window.
- StatusPanel.class – the StatusPanel is a user interface class. It is used to display messages or hints to the user. The StatusPanel is located at the bottom of the SchematicPanel.

- TextView.class – the TextView class is a user interface class. It is used to display the SPICE and Verilog views of a cell to the user. The TextView class optionally allows the user to add, edit, or delete the SPICE or Verilog view.
- UrlDialog.class – the UrlDialog class is a user interface class. It is used by the LibMgr class to input parameters necessary for loading cells from a network server.
- V.class – the V class is a PrimitiveCell class. It represents an independent voltage source. This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- Vcc.class – the Vcc class is a PrimitiveCell class. It represents a single-ended voltage source. The Vcc value is specified relative to ground (SPICE node 0). This PrimitiveCell class supports SPICE extraction, but not Verilog extraction.
- WebTop.class – the WebTop class is the top-level applet class. This is the executable class that should be run by appletviewer, or embedded in a web page.
- WebTopFrame.class – the WebTopFrame class is a user interface class. The WebTopFrame is the schematic editor window. It contains the ControlPanel, SchematicPanel, and StatusPanel.
- Wire.class – the Wire class is a PrimitiveCell class. Despite being lumped with the PrimitiveCell classes, it is not in fact a PrimitiveCell! The Wire class implements ideal wires. The Wire cell should not appear in either SPICE or Verilog netlists; the netlist extractor should automatically ignore Wire cells.

- WrapUtil.class – the WrapUtil class is a data structure class. It consists of a collection of useful network-related methods. The methods of WrapUtil are used to generate headers ad prepare data to be sent to network servers.
- XNOR.class – the XNOR class is a PrimitiveCell class. It represents a two-input xnor logic gate. This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.
- XOR.class – the XOR class is a PrimitiveCell class. It represents a single-input xor logic gate. This PrimitiveCell class supports Verilog extraction, but not SPICE extraction.

REFERENCES

[1] Adobe Systems Incorporated. *PostScript Language Reference Manual.* Addison-Wesley Publishing Company, 1985.

[2] J. Gosling and H. McGilton, "The Java Language Environment: a White Paper," *URL http://www.javasoft.com/doc/language_environment/*

[3] D. Lidsky, "PowerPlay: an Interactive Design Experience," *URL http://infopad.eecs.berkeley.edu/PowerPlay/*

[4] Henry McGilton and Mary Campione. *PostScript by Example.* Addison-Wesley Publishing Company, 1992.

[5] "WELD Project: Web-based Electronic Design," *URL http://www-cad.eecs.berkeley.edu/Respep/Research/weld/index.html*

[6] I. van Rienen, "Digital Simulator," *URL http://www.lookup.com/Homepages/96457/digsim/index.html*

[7] T. Xanthopoulos, "Pythia," *URL http://braves.mit.edu/pythia-doc/pythia.html*