

ACTIVE-CADTM

Real-Time Interactive CAE Tools

Logic Simulator

User's Guide

Seventh Edition

Revision 2



*Automated Logic Design Company, Inc.
3525 Old Conejo Rd. #111
Newbury Park, CA 91320
Phone (805) 499-6867
Fax (805) 498-7945*

Seventh Edition

Revision 2, January 15, 1996

COPYRIGHT

Copyright © 1985-1996 by ALDEC. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of ALDEC, Newbury Park, CA 91320.

DISCLAIMER

ALDEC makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, ALDEC reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of ALDEC to notify any person of such revision or changes.

TRADEMARKS

ACTIVE-CAD, Active Schematic, SUSIE and SUSIE-CAD are trademarks of ALDEC, Inc.

ACCESS, ACTEL, ALTERA, AMD, MACH, MAX, NEC, EPSON, LASER JET, HP, FUTURENET, OMATION, RACAL-REDAC, TANGO, CAPFAST, CAD-NETIX, CASE TECHNOLOGY, APPLICATION BRAVO, XNF, LCA, XILINX, ORCAD, P-CAD, IBM PC/XT/AT, PS/2, EDIF, P-DIF, VIEWLOGIC, VIEWSIM, WINTEK, MENTOR, POST SCRIPT, CORELDRAW are trademarks or registered trademarks of their respective holders.

This edition applies to Release 2.0 of the software and higher. It is a reference guide for the ACTIVE-CAD and ACTIVE-FPGA products

ALDEC

Automated Logic Design Company, Inc.
3525 Old Conejo Rd., Suite 111
Newbury Park, CA 91320
Phone (805) 499-6867
BBS (805) 498-4086
Fax (805) 498-7945
Email: support@aldec.com

Preface

The ACTIVE-CAD simulator is a real-time interactive design analysis tool. It can work both with an on-line schematic capture and with any number of off-line schematic editors. Since the design resides in a set of directly accessible data tables, you can interact with your design as if it were a real hardware breadboard.

To take full advantage of the ACTIVE-CAD power, you must remember how it differs from other popular simulators:

- ☐ you can start simulation the moment you load a netlist or draw a single gate; no compilation of the design or test vectors is needed
- ☐ all devices are electrically active the moment they are connected to active signal lines
- ☐ you can replace parts while the simulation is in progress and instantly view the improved design performance
- ☐ you can apply stimulus signal(s) or test vector(s) to any design section or device pin without any compilation
- ☐ all feature selections are instantly active, without compilations
- ☐ the simulator can directly control external hardware and respond to signals generated by the external hardware
- ☐ ACTIVE-CAD works like a real hardware breadboard

Manual Organization

The manual is comprised of five (5) logical parts:

- ☐ an introduction to real-time interactive simulation
- ☐ general description of ACTIVE-CAD menus
- ☐ simulation process
- ☐ simulation supporting tools
- ☐ appendices

An introduction to real-time interactive simulation

Since the real-time interactive simulators represent a new technology, *Simulation Basics* (Chapter 1) will allow you to become familiar with the real-time design analysis. Some of you may try your own designs after reading this chapter. However, we urge you to read the additional chapters to further explore the breadth of the ACTIVE-CAD simulator.

General description of ACTIVE-CAD menus

A general description of ACTIVE-CAD menus is provided in *Using The ACTIVE-CAD Simulator* (Chapter 2). Since this chapter gives you an overview of all available features, you should become familiar with it to get the most out of the software package. This is also the best product reference guide, right after the Index.

An outline of the simulation process

A detailed description of the simulation process starts with the section *Loading A Design* and ends with *Printing Simulation Results* in Chapter 2.

Simulation supporting tools

The simulation support tools are described in chapters *External Test Vector Editor*, *Mobic Model Builder* and *Simulator Macro Operations*. The VHDL Shorthand software product has its own manual. All these chapters describe some advanced test vector development products and modeling tools. Learning these tools may be optional.

Appendices

This is a collection of ACTIVE-CAD-related specifications which lists libraries, some application notes and specifications. If you ever need to write your own interfaces to ACTIVE-CAD netlists and test vectors, consult *Appendices* for all the necessary information.

Application notes on specific design issues

Some of the design issues have been covered in several sections of the manual but with different examples. Most design-specific customer inquiries have found their way into the section *Quick Application Notes* (Chapter 1). Additional application books are also available on key technologies.

Your comments are invaluable and we invite you to share your experiences with us. We will include them in additional sections and manuals. We are happy to answer all your questions to make your time with ACTIVE-CAD most productive.

Chapter 1

Simulation Basics

Introduction

There are many digital simulators. However, none of them will give you the hardware breadboard feeling like the real-time interactive ACTIVE-CAD simulator. Everything you do with ACTIVE-CAD will always be directly related to your hardware experience.

There are four (4) basic steps in design simulation:

- ☐ creating an electronic breadboard
- ☐ selecting test points for monitoring
- ☐ creating and applying design stimulus signals
- ☐ analyzing the simulation results

ACTIVE-CAD creates an electronic breadboard of your design directly from its netlist. However, unlike the hardware breadboard, your electronic breadboard is created within seconds and is perfect in every respect: no bad solder joints, miswirings, wrong polarities, bad chips, ground noise, signal line crosstalk, to name a few possible problems.

The electronic breadboard is tested with signals which are called test vectors. Each test vector lists logical states of all stimulus signals at a selected time interval. If you are testing a 2-input gate, you may only need four test vectors (00, 01, 10, 11). However, if you are testing an 8-bit binary counter, then you may need up to 256 test vectors (clocks) to test its operation.

Hardware breadboards allow you to apply signal generator probes to any pin on a live board. Similarly, ACTIVE-CAD test vectors can be cre-

ated and applied to any test point on the schematic while the simulation is in progress. Because of that, the ACTIVE-CAD electronic breadboard operates like a real hardware breadboard, and you can manually toggle any signal line in real time. This analogy between ACTIVE-CAD and real hardware is so strong that you need only a few hours to learn this simulation tool.

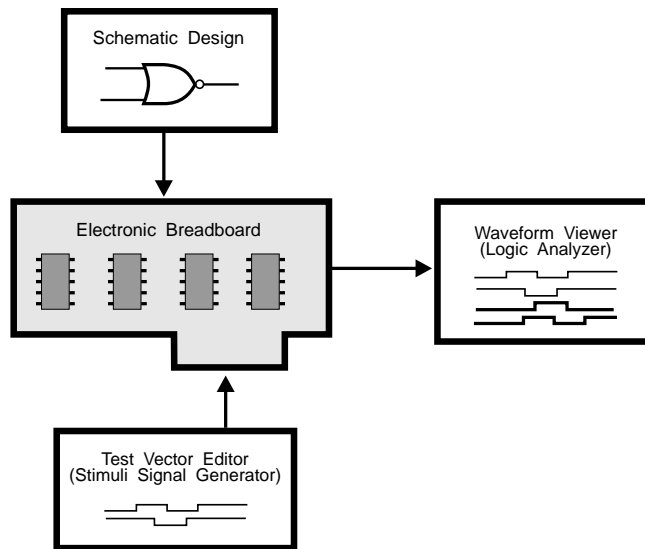


Figure 1-1. SUSIE Simulator Environment.

ACTIVE-CAD comes with a test vector editor that allows you to create signals for stimulating the design circuits with any desired action. You can apply these stimulus signals or test vectors at any test point in your design. The test vector editor can produce 1,000 independent signal waveforms (channels), each capable of a 100 GigaHertz clock speed. The ACTIVE-CAD signal generator is thus superior to any hardware signal generator, at a fraction of the cost.

Since ACTIVE-CAD can display 1,000 signal channels, each operating at a 100 GigaHertz clock speed, and since it automatically displays all circuit timing errors, it is more powerful than any logic analyzer.

The ACTIVE-CAD design environment is shown in Figure 1-1. Since the ACTIVE-CAD simulator behaves like a real hardware breadboard, ACTIVE-CAD-based designs can operate in a closed loop with the actual hardware located outside the computer.

Creating An Electronic Breadboard

Creating a working hardware breadboard requires someone to wire sockets and insert components. The same is true for the electronic breadboard. Since ACTIVE-CAD is an incremental design environment, it wires the electronic breadboard and inserts devices (simulation models) into sockets as you draw them. Loading an external netlist automatically creates the entire electronic breadboard, including loading of the appropriate device models.


Missing Models

If you are using some components for which ACTIVE-CAD does not have a model, such as the Pentium microprocessor, then the socket will remain empty. Also, if you have not purchased some libraries (not encoded into the keylock), their component sockets will remain empty despite that they have been drawn in the ACTIVE-CAD schematic editor and the model libraries are listed in the library manager.

The empty sockets in electronic breadboards behave exactly like empty sockets in hardware breadboards they don't load any input signals and they float all output signal lines. Since ACTIVE-CAD is a real-time interactive simulator, you can assign to the outputs of such devices your own test vectors that emulate their behavior and simulate the entire design despite the absence of these device models. You will find more information on handling missing models in the *Test Vectors* and *Handling Missing Models*, included in *Loading A Design* (Section of Chapter 2).

Entering Design Test Points From Schematic

Since ACTIVE-CAD schematic imitates real hardware breadboard, you can select tests points for on-line background simulator directly from the schematic. To select the test point selection mode, click on the **Simula-**

tion toolbox  icon in the schematic editor and then click on the desired test points. Each click will produce test points at pins and nodes or signal names. The selected test points will also be fed into the simulator once it is activated.

Loading External Design Netlists

To create an electronic breadboard of a design you need to load its netlist. Because ACTIVE-CAD is a universal design verification tool, it accepts netlists from other schematic editors. Over twenty of the most popular schematic netlist formats are available, including some formats which are hardly in use today. This means that you should be able to simulate a design no matter how old it may be.

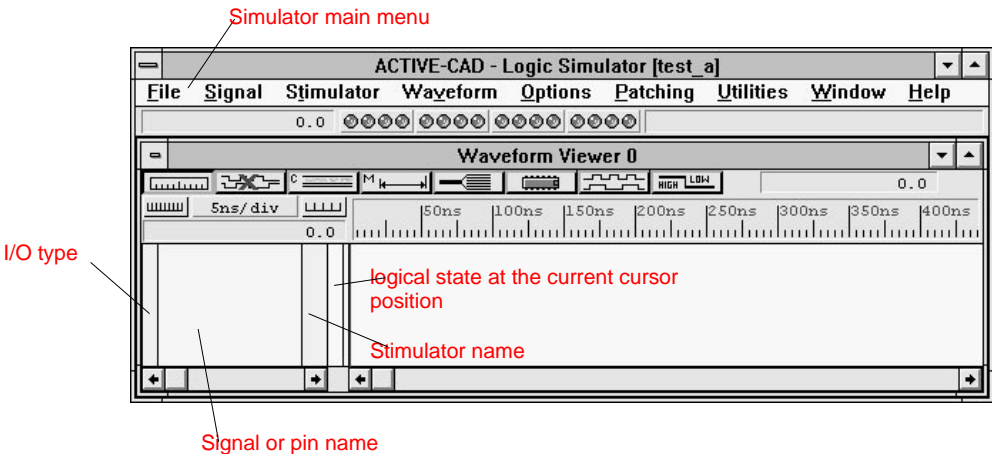


Figure 1-2. SUSIE Simulator main window.

Upon loading a netlist into the ACTIVE-CAD simulator, it will display the screen shown in Figure 1-2. Familiarize yourself with the terminology in this figure because it will be very useful in the following chapters. To select test points in design represented by imported netlist, use the procedure described in reference to Figure 1—7.

File	
Load Test Vectors...	Load Simulation...
Load ASCII Test Vectors...	Save Simulation...
Save Test Vectors...	Page Setup...
Save ASCII Test Vectors...	Print Setup...
Load Memory Chip...	Print...
Save Memory Chip...	Print Error Report...
Load Memory Block...	Project Manager...
Save Memory Block...	Project Libraries...
Load Fuse Map...	Load Netlist ...
Exit	Test PLD ...

Figure 1-3. File Menu Options.

To load a design netlist, select the **Load Netlist** option from the **File** menu shown in Figure 1-3. In response, ACTIVE-CAD displays the **Load Netlist** window in Figure 1-4. Select the desired netlist format from the **From Format** field. Some of the available netlist formats are shown in Figure 1-5.

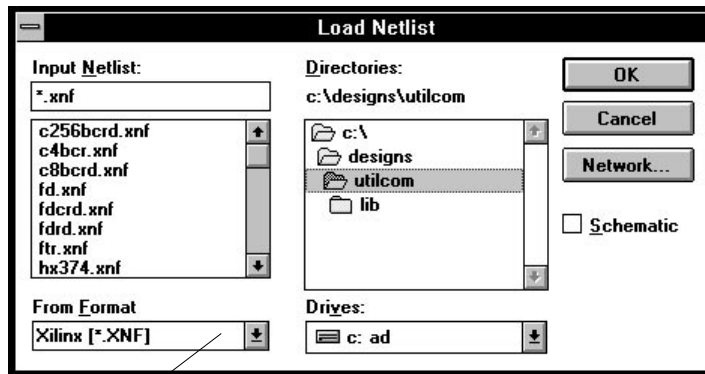


Figure 1-4. Load Netlist window.

Select the
desired
netlist format

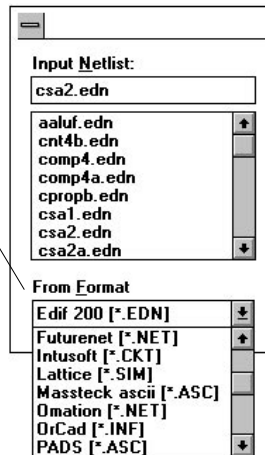


Figure 1-5. Simulator Netlist Formats.

ACTIVE-CAD automatically lists in the **Input Netlist** field (Figure 1-4) all netlists in the selected netlist format. This simplifies your search because only the desired netlist files are listed.

Example 1:

Select from the **From Format** field shown in Figure 1-5 the SUSIE Binary [*ALB] format. Click on the **TEST_A** project in the **Directories** window and then on the **OK** button. In response ACTIVE-CAD should list the **TEST_A.alb** netlist which is supplied with the software as an example of the flat netlist format. Click on the **TEST_A.alb** netlist and then on the **OK** button. ACTIVE-CAD loads the selected netlist and displays messages about the loading progress. Don't be surprised if after the netlist has been loaded there are no changes on the simulation screen; to avoid

screen clutter only the data you have explicitly requested will be displayed on the simulator screen.

For better understanding of the *TEST_A* netlist operation, refer to Figure 1-6, which shows the schematic design from which this netlist was generated. ACTIVE-CAD can provide pin connectivity information and can even recreate an entire schematic from a netlist. However, for quick reference use the readily available schematic in Figure 1-6.

The **Directories** window in the **Load Netlist** window (Figure 1-4) allows you to search for the netlist files on any drive and throughout the entire network if ACTIVE-CAD has been installed on one. Since these are typical Windows operations, no details on the file search is provided here.

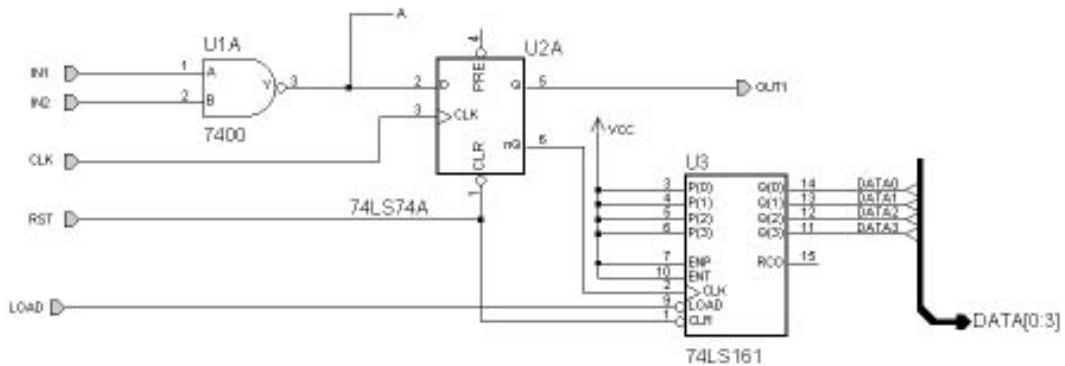
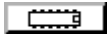


Figure 1-6. *TEST_A* Schematic Design.

The name of the currently loaded netlist appears in the title bar of the logic simulator. For example, Figures 1-2 and 1-3 show that the *XBL* netlist has been loaded into the simulator. An additional verification that the right netlist has been loaded can be performed by clicking on the **Signal Selection** button  in the **Waveform Viewer** toolbar. In response, the **Component Selection** window shown in Figure 1-7 will list signal names and components of the loaded design. A quick review of that list will confirm if you have loaded the desired design.

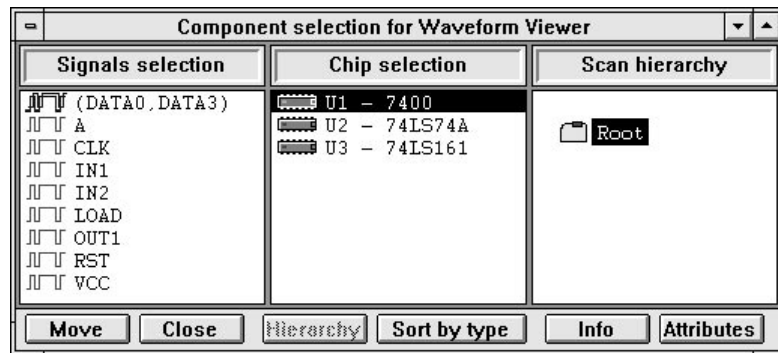


Figure 1-7. Design Component Display.

NOTE: If you are using a schematic editor that is integrated with ACTIVE-CAD through the Windows DDE interface protocol, then you do not need to load any netlist because all schematic changes are automatically implemented in the simulator data base. The most popular schematic editors that use the DDE protocol to communicate with SUSIE are SUSIE-CAD, ACTIVE-CAD, DIGILAB and Virtual Hardware Editor.

Selecting Design Test Points

To stimulate a hardware breadboard, you need to set up a hardware signal generator and apply its signals to selected test points on the breadboard. A similar process is employed with an electronic breadboard. However, its stimulus signals can be generated and applied with much greater ease.

To select design test points to which you want to apply external signals, select the **Add Signals** option from the **Signals** menu. If you have loaded the *TEST_A* netlist that comes with the ACTIVE-CAD simulator, then in response ACTIVE-CAD will display the window shown in Figure 1-7. The **Signals Selection** field displays all the design signals, such as input and output terminals, node names and buses. To select any of these signals for display, double-click on it with the left mouse button. This will instantly copy the selected signal into the **Waveform Viewer** window (**Signal** field in Figure 1-2).

Example 2:

With the *TEST_A* netlist loaded, double-click on the *IN1* input signal listed in the **Signal Selection** field of the **Component Selection** window (Figure 1-7). Note that the signal has been automatically copied to the **Signal** field of the **Waveform Viewer** window (Figure 1-2).

If you have a large number of adjacent signals to be transferred to the ACTIVE-CAD display, double-clicking may be tedious. Instead, you can first select the desired signals and then transfer them all at the same time. To select the signals, hold down the **[Ctrl]** key and click on each desired signal name. When all signals have been selected, click on the **Move** button in the **Component Selection** window (Figure 1-7). Note that all the selected signals have been transferred into the ACTIVE-CAD logic simulator window in the order in which they have been selected.

Example 3:

Click on the *OUT1* signal. Next, depress the **[Ctrl]** key and click in sequence on the *CLK* and *A* signals in the **Signal Selection** field of the **Component Selection** window in Figure 1-7. Following this, click on the **Move** button. Note that the signals have been transferred to the **Signal** field of the ACTIVE-CAD display in the order in which they have been selected.

You can also select any device pin as a test point and apply stimulus signal to it. To select a device pin, double-click on a selected device listed in the **Component Selection** window in Figure 1-7. In response, ACTIVE-CAD displays a **Pins for:** window that lists all the pins for the selected device (Figure 1-8). Double-click on the selected pins to move them in the desired order. Marking signals and then activating the **Move** button copies signals in the order in which they were listed in the window and not in the order in which they were selected.

Since ACTIVE-CAD is a real-time interactive simulator, you can add and delete any test points while the simulation is in progress. There is no limit on what you can select to the display screen and stimulate with signals developed in the simulator or loaded as a test vector file.

Example 4:

With *TEST_A* netlist loaded, double-click on the *U1-7400* in Figure 1-7. In response, ACTIVE-CAD displays Figure 1-8 which lists all the U1 device pins in the **Pins For: U1-7400** field. Double-click on the *A1-1* pin and see it moved to the ACTIVE-CAD screen.

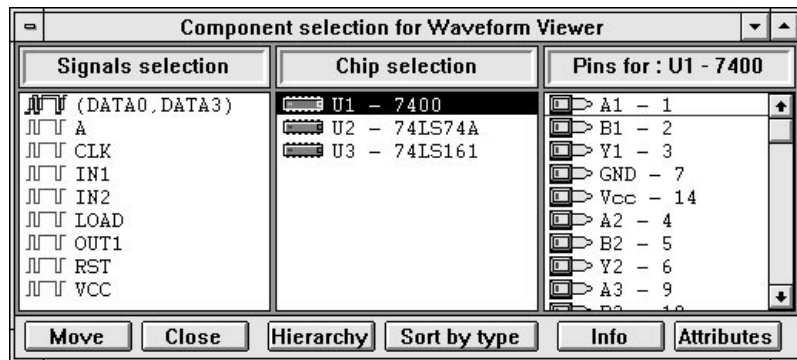


Figure 1-8. Display of U1 pins.

Continue to copy signals to the waveform viewer until the waveform viewer shows the signals listed in Figure 1-9. Save this setup of test points by selecting the **Save Test Vectors** option in the **File** menu. For consistency with this *Guide*, save it as the *TEST_A* file.

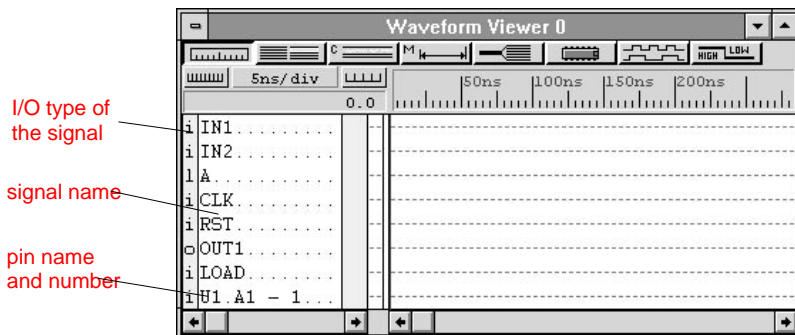


Figure 1-9. Selected design test points.

Selecting Test Points In Hierarchical Designs

If your ACTIVE-CAD has the hierarchical simulation option, you will be able to load hierarchical design netlists. The ACTIVE-CAD software comes with the TEST_H hierarchical netlist, which is based on the hierarchical schematics shown in Figures 1-10, 1-11 and 1-12. Loading hierarchical netlists is identical to loading a flat netlist, described in Example 1.

After loading the TEST_H1 hierarchical design netlist, select the **Add Signals** option from the **Signals** menu. In response, ACTIVE-CAD will display the window shown in Figure 1-13.

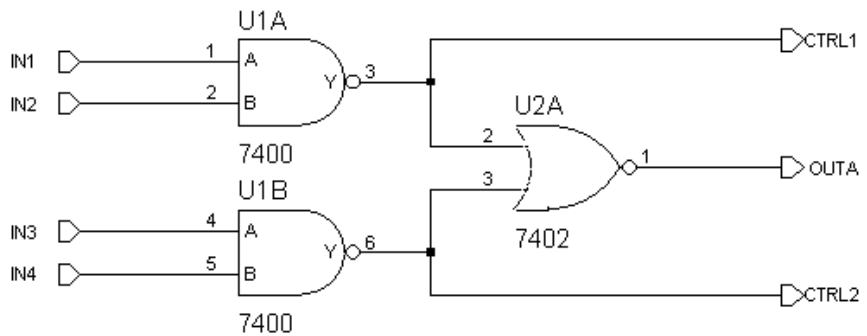


Figure 1-10. Schematic Macro H1.

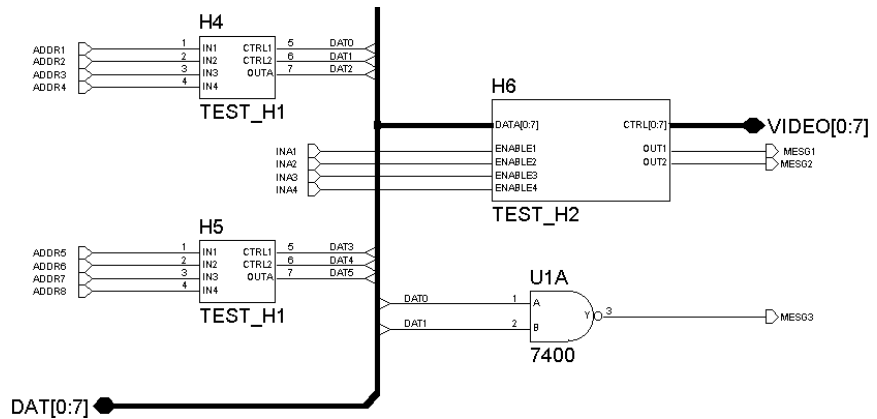


Figure 1-11. Schematic Macro H2.

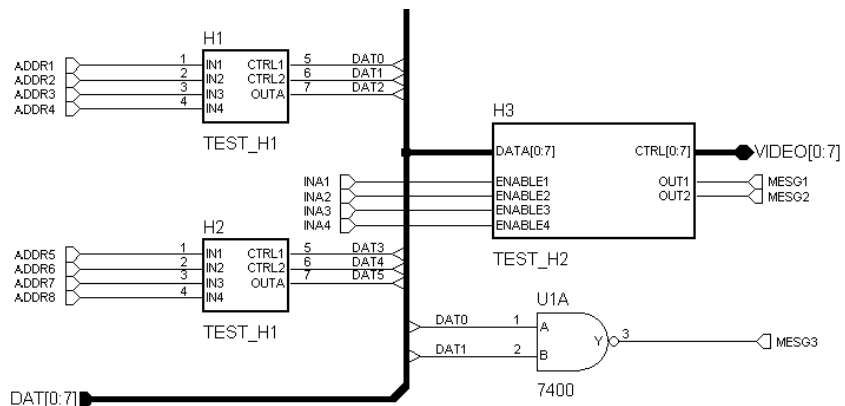


Figure 1-12. Top Level Schematic.

The **Signals Selection** field displays all the design signals at the selected hierarchical level. Since the starting point after loading a hierarchical netlist is the top level design, Figure 1-13 lists all signals at that level. To select any of these signals for the display, double-click on it with the left mouse button. This will instantly transfer the selected signal into the **Signal** field of the window shown in Figure 1-2. You can also hold down the **CTRL** key, click on each desired signal, and then transfer them all to the ACTIVE-CAD screen by activating the **Move** button.

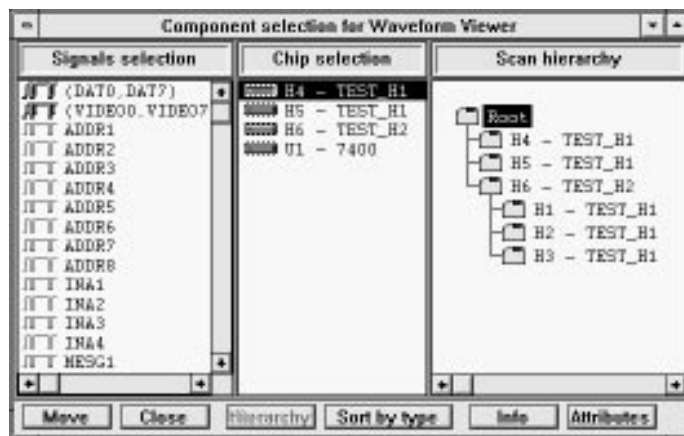


Figure 1-13. Hierarchical Design Structure Display.

To select signals from a selected hierarchical level, click on the selected hierarchical level in the **Scan Hierarchy** window shown in Figure 1-13. The **Signals Selection** window displays all signals and the **Component Selection** window lists all macros and devices at that hierarchical level.

You can select any signals and device pins for the screen display as described in reference to the flat design netlists.

Creating Design stimulus Signals

One of the main advantages of simulation over hardware breadboarding is that you don't have to do any logic analyzer setups. All data is automatically captured and analyzed by the simulator. This not only improves design quality but also saves you a lot of time. However, development of efficient test signals still remains a major challenge.

The incremental simulation process is very similar to hardware debugging, where each input signal line is fed signals from a separate signal generator channel. However, once all the hardware pieces have been individually debugged and integrated, they are tested with a set of signals that have a strictly predefined time relationships. These signal sets are called test vectors.

Since the design analysis needs change as the design evolves from a simple circuit to a fully operational system, the design testing methods must change as well. It is thus important to remember about the changing needs and to use the optimum analysis methods for each stage of design development:

- ☐ use primarily signal waveforms for incremental circuit development
- ☐ continuously optimize the signal waveforms as the design grows
- ☐ save the signal waveforms as ASCII or binary test vector files
- ☐ use test vector files exclusively for the final system verification

Since many designers have strong hardware breadboarding background, ACTIVE-CAD comes with a signal generator that mimics the operation of hardware signal generators. This simplifies the transition from hardware circuit debugging to incremental software design testing.

Since the final design testing has different objectives from those of incremental design analysis, *Simulator Macro Operations* (Chapter 5) will address test vector generation for batch mode design verification.

Signals that you use to stimulate the circuit are the basis for effective design testing. For this reason, you should become thoroughly familiar with effective signal waveform and test vector generation. Out of all the chapters in this manual, the one on test vectors is by far the most important and even if you are an experienced designer, you should still familiarize yourself with the enclosed figures, warnings and notes.

Signal Waveforms and Test Vectors

One of the key issues in design analysis is the use of signal waveforms and test vectors in different design stages. Figure 1-14 shows the difference between these terms in graphical form.

- ❑ Signal waveforms are the horizontal waveforms that start at time t_0 and continue to the time t_n . For example, the signal waveform IN1 is a string of signal logical values at times t_0, t_1, \dots, t_n (0,1,0,1,0,1...)
- ❑ Test vectors are the logical states of all signal lines at the given time instance. They can be viewed as vertical slices across all signal lines. For example the test vector TV0 has 001 logical value because IN1=0, IN2=0 and A=1 at the time t_0 (TV0=00H). Similarly, TV1=10H, TV2=01H, etc.

NOTE: The test vectors are counted from the top-most or *IN1* signal line in Figure 1-14 (the left-most character in TV0) to the A signal at the bottom (right-most characters in TV0). Also, the input signals have 0 for Low and 1 for High. Output signals have L for Low and H for High. Since A is an output signal, the TV0 test vector assigns to it H instead of 1. Similarly TV1 assigns to it L instead of 0.

A signal waveform gives you a good view of the selected signal behavior over time. The test vectors on the other hand give you the exact relationship between signals at any selected time instance. When we deal with such signals as clocks and override signals, we prefer to use signal waveforms. Also, when testing a design in real time, one would usually generate and apply a set of signal waveforms rather than a set of test vectors.

Some designers prefer to stimulate their designs with signal waveforms. Others prefer to use test vectors. The truth is that you need both methods because the design needs change as you move from the incremental debugging mode to the final design verification. If the signal waveforms have been continuously upgraded and optimized during the incremental design analysis, then test vectors will readily be available for the final design verification.

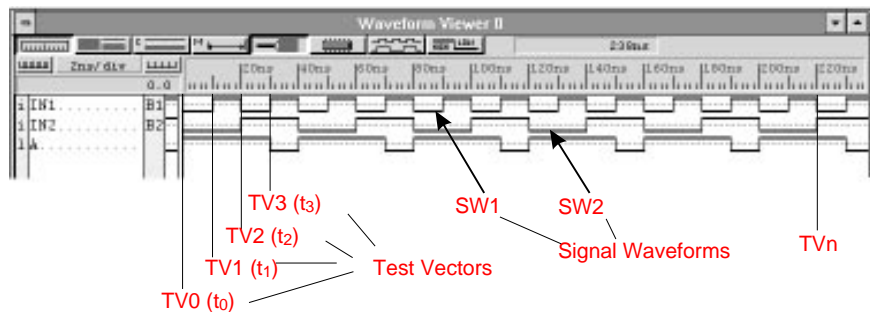


Figure 1-14. Difference between waveform & test vector.

ACTIVE-CAD comes with:

- ☐ a signal waveform editor
- ☐ an off-line (external) test vector editor
- ☐ a test vector macro editor

Ready-Made Signal Waveforms

Designing signal waveforms is often very cumbersome and time consuming, particularly for new and complex designs. To make your work easier, ACTIVE-CAD comes with over fifty (50) ready-made signal waveforms that can fulfill over 80% of your interactive design testing needs.

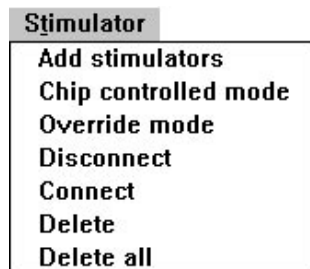


Figure 1-15. Stimulator menu.

To differentiate between signal waveforms and test vectors, we will call the signal waveforms stimulators. To allow for a broad range of operations on the stimulators, a special **Stimulator** sub-menu (Figure 1-15) is provided within the Logic Simulator main menu:

- ☐ The **Add Stimulators** option displays a window (Figure 1-16) with ready-made stimulators. If you assign them to signal names on the ACTIVE-CAD screen, they will directly control these signal lines.

- ☐ The **Chip Controlled Mode** gives direct control over the selected signal line to the chip output without deleting the stimulator which has been previously controlling that signal line
- ☐ The **Override Mode** allows the stimulators to override the assigned output pin signals
- ☐ **Disconnect** eliminates any effect of the assigned stimulator
- ☐ **Connect** restores the effect of the assigned stimulator
- ☐ **Delete** completely deletes the selected stimulator from the screen
- ☐ **Delete all** deletes all stimulators from the screen

How Use Ready-Made Stimulators

The ACTIVE-CAD stimulators are shown in Figure 1-16. They have pre-assigned names which you can assign to signal names displayed on the screen. To assign a stimulator to the selected signal line:

1. Click on **Add Stimulators** in the **Stimulators** menu.
2. Click on the selected signal name, e.g. *IN1* shown in Figure 1-14.
3. After the selected signal name turns blue, click on the desired stimulator shown in Figure 1-16.
4. The selected stimulator name appears in the **Stimulator** column, next to the signal line.

You can also assign stimulators as follows:

1. Click on a stimulator in Figure 1-16
2. Holding the left mouse button down, drag the cursor to the desired signal name, e.g. *IN2* in Figure 1-14.
3. Release the mouse button. The selected stimulator is instantly assigned.

The buttons in the **Stimulator Selection** window of Figure 1-16 are arranged into six (6) groups:

- ☐ **Keyboard keys** (A-Z) which can be used for direct interactive toggling of signal lines while the simulation is in progress.
- ☐ A **Software-emulated 16 bit counter** with TRUE (B0-B15) and INVERTED (NB0-NB15) outputs. These outputs provide clock signals with a 50% duty cycle. The clock of the counter is adjustable.
- ☐ **Formula stimulators.** Formulas allow you to generate custom-made signal waveforms from nested expressions that involve signal values

and their duration. These formulas can be assigned to selected **Form** buttons (F0-F15) which can then be assigned to control signal lines.

- ☐ **Asynchronous clock** stimulators defined with the Clock Editor.
- ☐ **Control buttons** which allow you to control the signal overriding capability.
- ☐ **Formula and Clock Editor** which allows you to define the formula stimulators and asynchronous clocks.

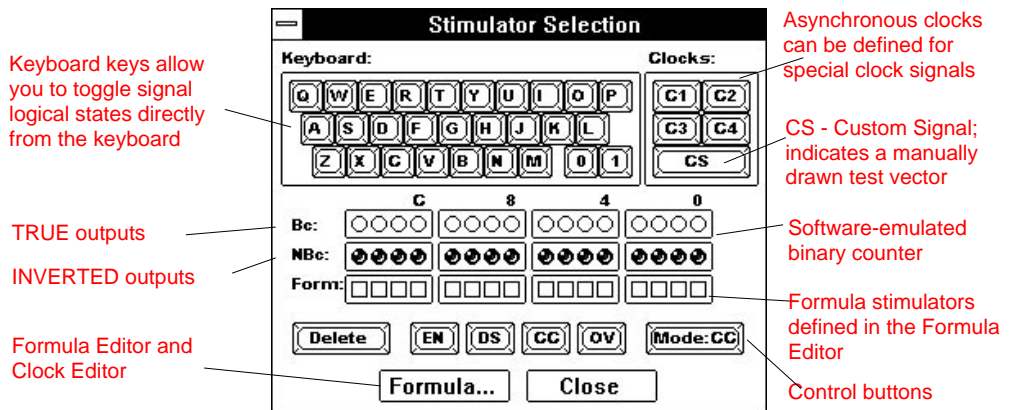


Figure 1-16. Stimulator Window.

Keyboard Keys

When assigned as stimulators, keyboard keys operate as SPDT switches that switch the signal line between Vcc and GND. Figure 1-17 shows keyboard key [A] assigned to the gate U3A B input and keyboard key [F] assigned to the reset input of the flip-flop U1A. Both keyboard keys directly control these inputs, irrespective of other signals that may be provided via connections to other sections of the design. Note that the [F] key cuts off the signal line connected to the other circuits. It is equivalent in hardware to cutting a wire between the flip-flops reset input and the other circuits on the board.

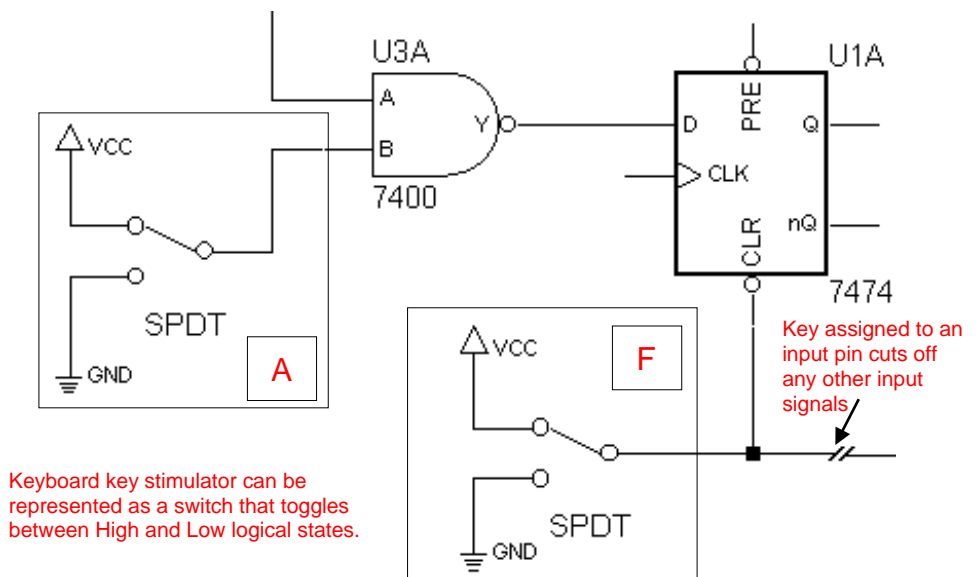


Figure 1-17. Keyboard key stimulators.

Keyboard keys are primarily used for direct toggling of selected signal lines while the simulation is in progress. For example, the circuit in Figure 1-18 is under direct control of the keyboard keys A, B, C, D, E and F.

Overriding device input pins

The device input pins are overridden by the assigned keyboard keys at all times. For example, referring to Figure 1-18, the [A] and [B] keys control the reset lines of U1A and U2B flip-flops, respectively. The [D] key controls the D-input of the first flip-flop.

Even if there is a circuit-generated signal fed into a device input, the keyboard key will always override such a signal. A good example of such a case is the [E] keyboard key which overrides the U1A-Q signal at the U2B-D input (Figure 1-18).

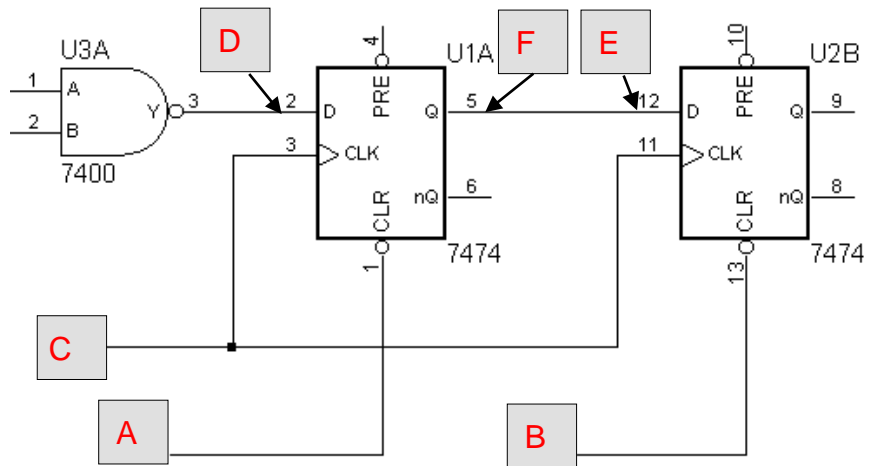


Figure 1-18. Direct Circuit Control with Keyboard Keys.

Overriding device output pins

The keyboard keys override device outputs automatically by forcing the **Override mode** (OV button in Figure 1-16 or **Override Mode** in Figure 1-15). All signals operating in the **Override mode** are red. To allow chip outputs to take control of its outputs, select the **Chip Control mode** (CC button in Figure 1-16). For example, the keyboard key [F] automatically overrides the U1A-Q output signal until the **Chip control mode** (CC button) is activated.

You can assign the same keyboard key, e.g. [D] key, to as many signal lines as needed. Using the options in the **Stimulator** menu shown in Figure 1-15, you can enable (**Connect** option) and disable (**Disconnect** option) the assigned keyboard keys.

Overriding net signal names

If a keyboard key is assigned to a netlist name (e.g. [C] in Figure 1-18), it controls all the signals in the node if it is set to the overriding mode. Otherwise, any active output pin in the node will control that node.

Binary Counter

ACTIVE-CAD comes with an in-software implemented 16-bit binary counter. It is shown in Figure 1-19. The counter automatically provides 16 TRUE (B0-B15) and 16 INVERTED (NB0-NB15) stimulators that can be assigned to any signal line in your design. These clocks have a 50% duty

cycle. The binary counter is driven by the CLK signal, which is set by selecting the **Clock Settings** option from the **Options** menu. Note that you must enter one-half of the desired clock period. For example, if you want a 100 MHz clock speed (10 nanosecond period), you must enter 5 nanoseconds, or 1/2 of the clock period.

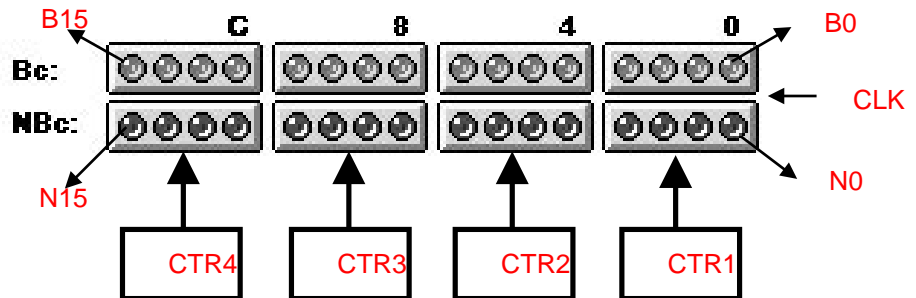


Figure 1-19. In-Software 16-bit Binary Counter.

You can rescale the binary counter input clock from 1000 Hertz to 100 GigaHertz. Each counter stage divides the clock in half. For example, if you have set the input clock at 100 MHz, the B0 output =100MHz, B1=50 MHz, B2=25 MHz, etc.

Each stage of the 16-bit binary counter drives its own LED lamp as shown in Figure 1-19. The B0-B15 signal lines are the TRUE binary counter outputs and the associated lamps are green. The inverted counter outputs (N0-N15) have red lamps.

You can assign the same binary counter output signal, e.g. B1, to as many signal lines as needed. The binary counter lamps are shown at the top of the simulator screen shown in Figure 1-2. They allow you to view the current state of this signal generator.

Formula Stimulators

The Formula Editor allows you to develop signal waveforms quickly and efficiently. Because these waveforms, also called formula stimulators, may be of any shape or duration, you can use them for non-repetitive functions and bursts of pulses of any duration. You can also use the formulas for designing clocks with special duty cycles and various logical states.

The formula stimulators give you the freedom to define and redefine signal waveforms while the simulation is in progress. This extremely simple yet very powerful method of designing the most complex design stimulus is being widely used by FPGA and board-level designers.

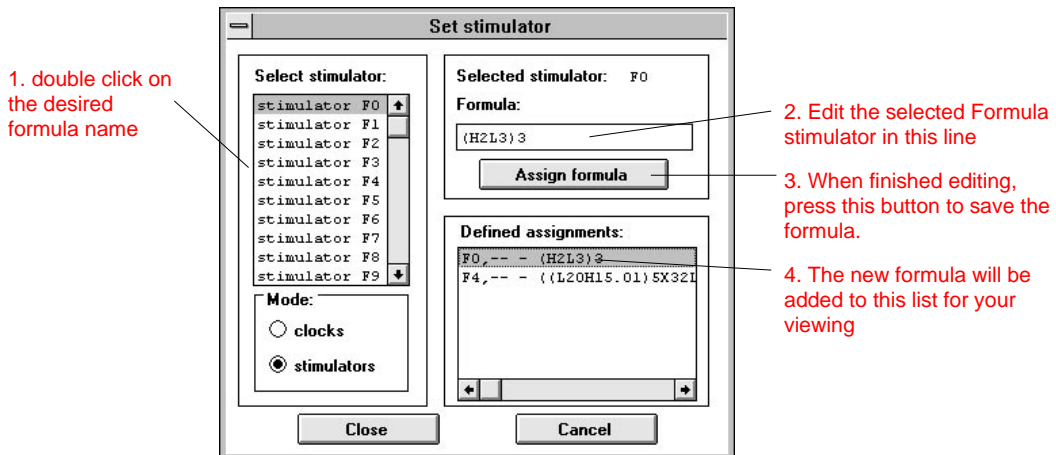


Figure 1-20. Formula Editor Window.

The process of defining a formula is comprised of writing:

- ☐ parentheses (used in nested expressions)
- ☐ logical signal state symbols (H,L,Z,X)
- ☐ duration of the signal states (e.g. 12.31 nanoseconds)
- ☐ the number of repetitions of the signals in parentheses
- ☐ brackets ([]) for defining buses

Example 5:

(H2L3)3

There is no limit neither on the depth of signal nesting nor on signal duration.

To activate the **Formula Editor**, click on the **Formula** button shown in Figure 1-16. In response, ACTIVE-CAD displays the window shown in Figure 1-20.

First, select the formula name from the **Select Stimulator** window by double-clicking on the selected name, e.g. F0. ACTIVE-CAD displays the selected name in the **Selected Stimulator** field.

Next, enter the signal formula in the **Formula:** window, e.g. ((L20H15.01)5X32L24)52. Remember that you must have the same number of left and right parentheses. Otherwise ACTIVE-CAD will warn you about an error.

After the signal formula entry is completed, press the **Assign Formula** button shown in Figure 1-20. Note that the formula has been transferred into the **Defined Assignments** field. This formula can be instantly assigned to any signal line on the ACTIVE-CAD screen. Click on **Close** to exit the window.

To assign a formula to a selected signal waveform, click on the signal name shown in Figure 1-14. The selected signal name turns blue. Invoke the **Stimulator Selection** window shown in Figure 1-16 and click on the Formula- associated lamp in the **Formula** field. You can also first click on the formula lamp in **Stimulator Selection** window and drag the lamp over to the selected signal name on the Logic Simulator screen.

There is a total of 16 formulas (F0...F15) provided with each ACTIVE-CAD simulator. However, you can request a special option which provides up to 256 signal formulas. When properly used, the signal formulas can generate test vectors faster than the most powerful test vector editors.

Graphical Waveform Editor

Since ACTIVE-CAD is a real-time interactive simulator, you can perform an on-line what-if analysis. Such analysis is very useful because you can override any input and output with one of the fifteen (15) logical states and emulate the desired signals at these test points. For example, you can emulate proposed FPGA changes by forcing the selected inputs and outputs to the desired or mentally computed logical states. By monitoring their effects at internal (chip) and external (system) levels, you know beforehand what the proposed design changes will do.

One of the best ways to force the desired logical states is the graphical editor. It allows you to tweak each signal with great precision and force the device pins to one of the fifteen (15) logical states. Resimulation and comparison of the new data with the previous ones may encourage or discredit the proposed design changes. In either case, you are much

closer to the problem solution, and for this reason the dynamic what-if analysis is the most outstanding feature of the real-time simulators.



Figure 1-21. Waveform Menu.

To invoke the graphical test vector editor, select the **Edit** option from the **Waveform** menu shown in Figure 1-21. In response, ACTIVE-CAD displays a set of 15 buttons with the different logical states (Figure 1-22). Note that the mouse cursor also changes (Figure 1-23).

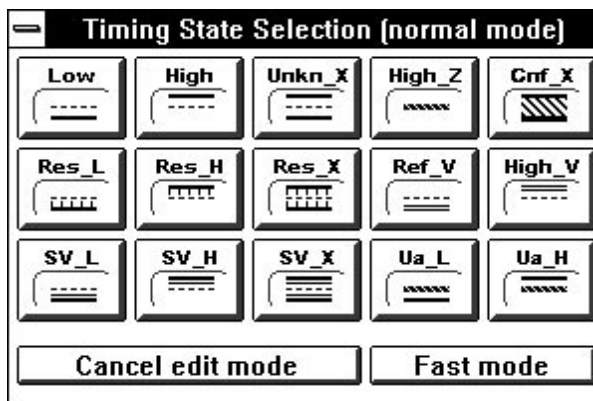


Figure 1-22. Logical States for Waveform Editing.

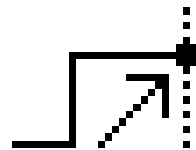


Figure 1-23. Editing Cursor.

List of logical states

The fifteen (15) buttons in Figure 1-22 represent the following signals:

- ☐ **Low** is a strong Low logical level
- ☐ **High** is a strong High logical level
- ☐ **Unkn_X** is a strong Unknown logical level
- ☐ **High_Z** is a weak 3-state state that can be overridden by weak signals
- ☐ **Cnf_X** indicates the presence of two strong signals of different logical values in the same node, e.g. High and Low logical levels
- ☐ **Res_L** is a weak Low logical level
- ☐ **Res_H** is a weak High logical level
- ☐ **Res_X** is a weak Unknown logical level
- ☐ **Ref_V** is a special logic level signal, such as used with ECL devices
- ☐ **High_V** is a special high voltage level such as used in line drivers
- ☐ **SV_L** is a strong power supply signal which overrides other signal
- ☐ **SV_H** is a strong power supply signal which overrides other signals

- ☐ **SV_X** indicates a conflict between two power signals in the same node
- ☐ **Ua_L** indicates a Low signal of undetermined strength
- ☐ **Ua_H** indicates a High signal of unknown strength

Follow these steps to edit signal waveforms:

1. Invoke the waveform editor (Figure 1-22).
2. Place the cursor at the desired signal waveform location and click the mouse button.
3. The associated signal name turns green and a blue vertical cursor appears at the selected waveform location.
4. Click on the desired logical state button shown in Figure 1-22.
5. Note that the signal waveform has changed to the new state; this change has taken place between the last signal transition and the current blue cursor location.
6. To exit the editor, click on the **Cancel Edit Mode** button in Figure 1-22.

Note that as you edit signal waveforms, the modified segments change to green. Also, a **CS** (Custom Signal) symbol has been forced into the **Stimulator** column to indicate that the displayed waveforms are being forced on the device pins. If the **CS** symbol appears at the device output, you need to select the **Override** mode by clicking on the **OV** button shown in Figure 1-16. Otherwise, the device will control its own output signal, despite the presence of the **CS** symbol.

If you need more precise waveform editing than the current time scale allows, place the cursor over the scale and press the mouse button. Note that a blue stripe is drawn as you drag the cursor over the area that you want to expand. When you release the mouse button, the waveform expands over the entire timing screen and the blue line disappears.

Applying Signal Waveforms

Read carefully the following rules on application of external stimulus to the circuit designs. There are only four (4) rules that govern the signal distribution in a node:

- ☐ A signal applied to the device input pin exists only at that input pin and does not spread through the node to any other pin. For example, if you apply a signal at U1A-2 (Figure 1-24), it will only control that particular input and will not have any effect on U1B-4.

- ❑ If you apply a signal at the device output pin, it must be asserted with the **Overdrive Mode** (OV button in Figure 1-16); Otherwise it has no effect on the pins and signal lines in the node. For example, feeding a signal into pin U1A-3 has no effect on any pin in the node unless you set this signal into **Override Mode** (see **Keyboard Keys-Overriding device output pins**, in this Chapter)

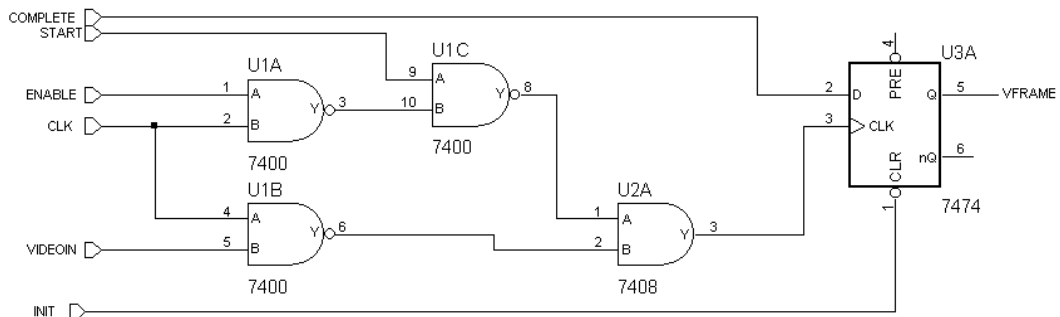


Figure 1-24. Applying Stimulators to Circuits.

- ❑ If you apply a signal to a node and the node does not have any device output pins (e.g. signal *CLK* in Figure 1-24), then it will control all pins in the entire node. However, if you apply a signal to a node (e.g. *VFRAME* signal name in Figure 1-24) which has any active device output, e.g. U3A-5, then it will be overridden by that output. Use node names primarily to connect wires between pages and to monitor what is going on in the node. However, apply stimulus signal to a node name only if there are no output pins in the node.
- ❑ If you apply a signal to an input terminal, e.g. *START*, it will control the entire node because there is no device output pin in the node. If the device output pin is tri-stated, there is no conflict with the signal applied at the terminal. However, if the device output is a totem-pole or active 3-state, it will be in conflict with the terminal applied signal because the terminal is treated as a totem-pole signal source.

Signal Waveforms Summary

- ❑ Use the keyboard keys to isolate design sections or to disconnect feedbacks and closely control problem areas. The precision of the keyboard key toggling is limited by the length of the **Short** step, and you can make it as small as 10 ps.
- ❑ Use the binary counter outputs in place of clocks as inputs to multiplexers, decoders and sections that operate on symmetrical signal waveforms

- ☐ Use formula signal waveforms for semi-random or asymmetrical signals, pulse bursts, etc.
- ☐ Swap signal waveforms as needed by selecting the appropriate buttons in the **Stimulator** window.
- ☐ Start with keyboard keys and progress to the more advanced and complex signal waveforms as the design becomes free of basic problems.
- ☐ All signal waveforms can be saved as test vectors. Save these test vectors, even if you are not sure of their future need. Each test vector should have ample comments of why it was used and what has been tested with it. These comments can be written directly into each signal waveform.
- ☐ Create an extensive library of test vectors by design sections and design functions; the quality of these libraries will help you to deal quickly with new problems and they will differentiate you from your less experienced peers.
- ☐ To conserve disk space, save all test vectors in binary format. ACTIVE-CAD can always convert these files to ASCII format when needed.

Most important, before you dive into the more intricate simulation issues, practice all four forms of signal waveform editing for at least 30 minutes.

ACTIVE-CAD comes with some advanced test vector editors (*External Test Vector Editor* and *Test Vector Macro Editor*) which you should practice after you have become experienced with the signal waveform editors.

Analyzing Simulation Results

ACTIVE-CAD automatically checks every pin of every device during each clock cycle for timing violations and bus conflicts. It is like having a logic analyzer with thousands of active signal channels. This logic analyzer is also quite intelligent because it knows what's right and what's wrong and even if you have not selected some signal lines to the display, ACTIVE-CAD will anyhow check them and report any problems.

To make the design analysis most effective, you should familiarize yourself with all the available options, ACTIVE-CAD comes with over forty (40) utilities that speed the design analysis and make your work easier. These utilities are located within the toolbox shown in Figure 1-25 and the **Patching**, **Options** and **Utilities** menus.

Design analysis is quick and effective if you follow these simple rules:

- ☐ Do your design analysis incrementally; never wait for the complete design because its size and complexity may overwhelm you.
- ☐ First, verify the functional design behavior.
- ☐ Next, using the unit delay or glitch simulator, check for race conditions.
- ☐ Set a plan for the worst-case design analysis; write it down and use it as a framework for timing analysis.
- ☐ Make sure that you have accounted for the temperature, supply voltage and loading effects.
- ☐ Save all design and test vector files; use ample comments in your test vectors.
- ☐ Most important - do not think of your design as schematic sheets; Rather look at it through the test vector files. You should spend at least as much time on design analysis as on creating the schematics and/or VHDL design itself.

Functional simulation mode

Each design verification starts with functional behavior analysis. By default, ACTIVE-CAD comes up in functional mode. However, if for some reason the central button of the toolbox shown in Figure in Figure 1-25 does not show the **FN** (functional) mode, click on it till it displays this symbol.



Figure 1-25. Simulation toolbox.

The power of the functional analysis mode lies not in its accuracy but in the simplicity of visualization because it directly shows the cause and effect relationship. For example all outputs of the gates are lined up with their inputs (Figure 1-26) and all flip-flop outputs are lined up with their rising or falling clock edges (Figure 1-27). Such signal waveforms are easy to analyze, and they allow you to quickly verify the functional behavior of your design.

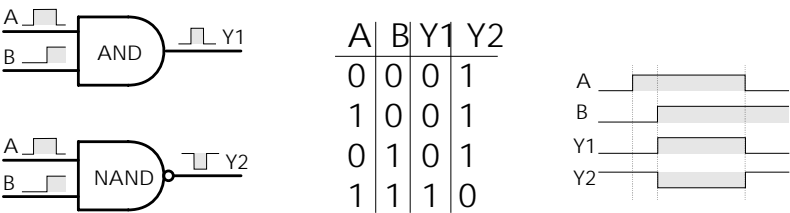


Figure 1-26. Functional Gate Simulation.

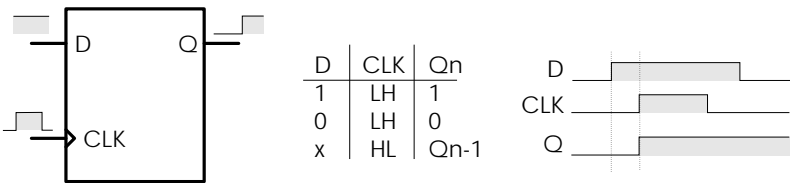


Figure 1-27. Functional Flip-Flop Simulation.

Even if the circuit you are designing seems simple enough to skip the functional simulation mode, do not take the risk. You would be surprised how many details may escape your attention even in the simplest designs. Once you have confirmed the functional design, you will be able to proceed to the other design phases with greater confidence.

Figure 1-28 shows typical waveforms for the circuit shown in Figure 1-6. Note how the OUT1 signal, which is produced by the U2A flip-flop, is aligned with the input clock CLK. A similar alignment is present between the U3 counter output Q0 and its input CLK signal.

Expanding the scale: If the alignment between signals is obscured by the display scale resolution, place the cursor over the scale, press the mouse button, drag it over the area that you want to expand and then release. As you drag the mouse cursor, a blue stripe appears marking the area selected for the scale expansion. Upon the release of the mouse button the marked area will be expanded to cover the entire screen.

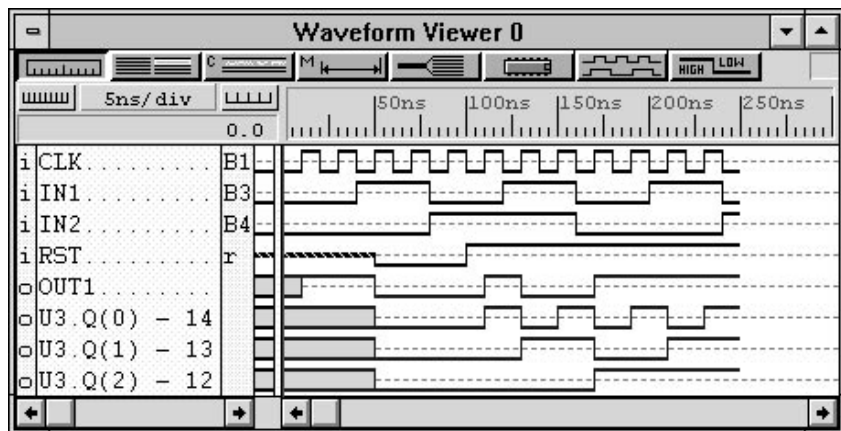


Figure 1-28. Functional Circuit Simulation.

Another way to expand and contract the display scale is to click on the scale adjustment buttons shown in Figure 1-29. Clicking on the **LSR** or **Lower Scale Resolution** button lowers the scale resolution, e.g. from 2 ns/DIV to 5 ns/DIV. Clicking on the **ESR** or **Expanding Scale Resolution** button will increase the scale resolution, e.g. from 5 ns/DIV to 1 ns/DIV.



Figure 1-29. Simulation scale adjustment buttons.

NOTE: If the functional simulation produces outputs with no cause, switch to the GLITCH simulation mode for viewing race conditions.

Most functional simulators simulate combinatorial and sequential devices with zero propagation delays and as a result they cannot warn you about any race conditions. Other functional simulators like SUSIE 4 and SUSIE 5 calculated the functional circuit behavior based on unit propagation delays but displayed the output data as if the circuits had zero propagation delays. This method had several advantages because it allowed you to capture race conditions in the functional mode. However, the output test vector files were not compatible with data from other simulators.

To generate test vector outputs that are compatible with other simulators, the current release of ACTIVE-CAD assigns zero propagation delay to combinatorial logic (gates, multiplexers, decoders, etc.) and unit propagation delay to sequential logical devices such as flip-flops, registers and counters. If these sequential logical devices have feedbacks through combinatorial logic, they may show output transitions that are unexplainable by a pure functional analysis. This is a rare case, but should it happen, switch the simulator to the GLITCH mode, which will display in detail the effects of the unit propagation delays.

Glitch Simulation Mode

When a functional simulator produces output signal waveforms for no apparent reason, you need to painstakingly analyze all circuit race conditions or use the slower and more complex timing simulator to find the reason for the unusual circuit behavior.

Since glitch simulators display all details that are hidden in the functional mode, they are excellent analysis tools of unknown circuit behavior. By assigning and displaying unit propagation delays for all components in all signal paths, these simulators automatically display signal spikes that cause the unusual circuit behavior.

To select the GLITCH mode, click on the **MODE** button shown in Figure 1-25 till it displays the **GL** letters. Simulate the design with the same test vectors that you have used for the functional simulation. The proper use of the glitch mode is as follows:

- ☐ Simulate your design in the functional mode
- ☐ If you find unexplained circuit behavior, switch the simulator to the GLITCH mode
- ☐ Load the test vectors that you have used for the functional simulation
- ☐ Simulate again and check for race conditions

You don't need to use the GLITCH mode if the outputs generated by the functional simulator look OKay. The GLITCH mode should be used only for quick and localized spike display. For example, if you simulate a functional circuit with 1,000 to 10,000 test vectors, you would use the GLITCH mode only with 5 to 20 test vectors, just enough to gain a good understanding of how some unexpected outputs are generated due to spikes on the selected signal lines.

For best results, rescale the signal waveforms to achieve a display similar to the one you had in the functional mode. Next, compare the sequential device outputs with the corresponding waveforms generated by the functional simulation. If the outputs have new states which have not been seen in the functional mode, the design has a race condition. However, if no new logical states have been detected on the sequential device outputs, there is no circuit race condition. At times you may see some spikes on signal lines that are the result of propagation delays. However, if they don't trigger any sequential circuits they are harmless and can be ignored, unless cross-talk and other effects must be considered.

Example 6:

Load the *TEST_C* netlist file, which is in the ALDEC format. The schematic from which this netlist was generated is shown shown in Figure 1-30. Next, load the signals and stimulators shown in Figure 1-31.

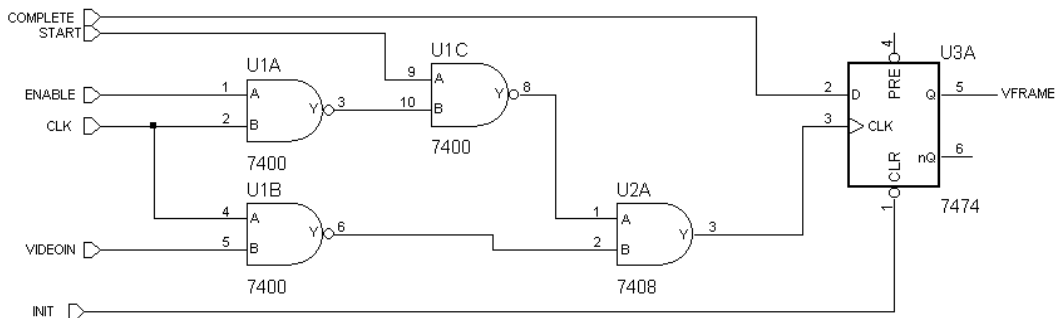


Figure 1-30. Assignment of Stimulators to Input Pins.

Toggle the *INIT* input signal line to Low by toggling the [I] keyboard key and clicking on the **STEP** button. This resets the flip-flop U3A shown in Figure 1-30. Next, set the *INIT* signal High, set the simulator to the **FN** functional mode and click on the **LONG** step button six times. Note that

the output signal U3A.Q1-5 is being set high without a rising clock edge at the U3A.CLK1-3 input.

To investigate why the U3A.Q1-5 output changes without a clock edge, let's switch to the GL GLITCH mode by clicking on the **MODE** button shown in Figure 1-25. Next, click the **LONG** step button eight times and watch what happens on the U3A.CLK1-3 signal line when the U3A.Q1-5 is changing to logical High. As shown in Figure 1-31, there is a small spike on the clock line. This signal spike is the result of the CLK signal line propagating through two different propagation channels (U1A-U1C and U1B) at the input to the U2A gate. By putting additional probes on these channels, you can view them in greater detail.

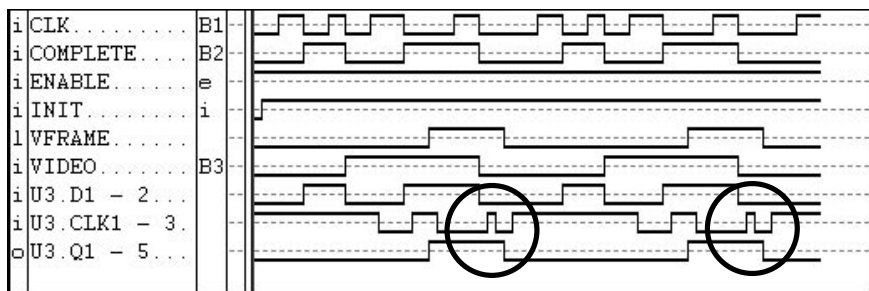


Figure 1-31. Display of Glitches on the Clock Pin.

Timing Simulation Mode

To have any confidence in a design, it must be simulated in the timing mode. To select the TIMING mode, click on the **MODE** button shown in Figure 1-25 till it displays the **TM** letters. Generally, you should first simulate the design with the same test vectors that you have used for the functional simulation. This will give you an instant confirmation that the previously simulated design will also pass the timing criteria.

To make sure that no design error goes undetected, ACTIVE-CAD checks every device model during each clock cycle. The models are tested both for timing violations and for bus conflicts. ACTIVE-CAD checks for the following timing violations:

- ☐ Setup and hold times
- ☐ Clock High and clock Low restrictions
- ☐ Pulse width of reset, preset, etc.
- ☐ Clock edge to reset pulse

- ☐ Reset pulse to preset pulse

All ACTIVE-CAD libraries have been written in VHDL Shorthand, which is an easy-to-use VHDL subset. The models have timing information but if you set the simulator to the functional, glitch or unit delay modes, then the models will not be analyzed for timing performance.

Example 7:

To become more familiar with the timing simulation process and available options, load the *TEST_A* netlist, which has been generated from the schematic shown in Figure in Figure 1-6. Select the signal names listed shown in Figure in Figure 1-32 and assign to them the stimulators listed in the same figure.

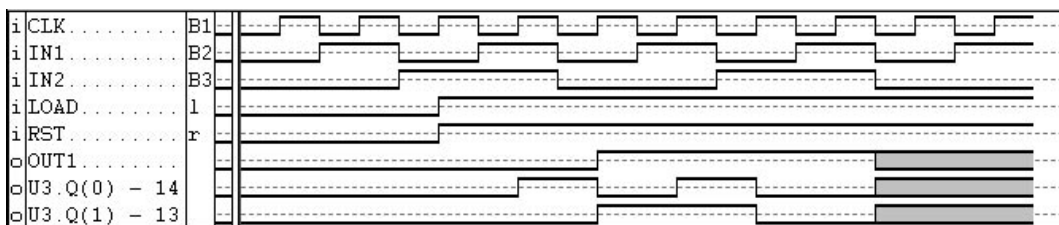


Figure 1-32. Timing Mode Circuit Analysis.

To better understand the timing simulation mode, follow these steps as part of *Example 7*:

1. Select the functional (**FN**) simulation mode.
2. Set the [l] and [r] keyboard-driven stimulators to the Low logical level and perform a single simulation step by clicking on the **STEP** button.
3. Toggle the [l] and [r] stimulators to the High logical state so that the flip-flop and counter can operate properly.
4. Click on the **LONG** simulation step button within the toolbox. Note that the functional simulation proceeds without any problems.
5. Click on the **MODE** button in the toolbox and switch the simulator to the **TM** (timing) simulation mode.
6. Click on the **STEP** button. Note that ACTIVE-CAD displays several timing violations warnings.
7. Click several times on the **OK** button within the error message window to familiarize yourself with the error messages. Click on the **Cancel** button to end the error message display.

8. As you can see from the error messages, the U2 flip-flop has some timing problems. Fortunately, ACTIVE-CAD is a real-time interactive simulator and you can instantly replace parts with faster ones and eliminate any timing problem that may exist:
9. Click on the **Patching** menu and select **Change Technology**. In response, ACTIVE-CAD displays Figure 1-33.
10. Double-click on U2-74LS74A in the **Chip Selection** field.
11. A new **Technology Selection** window appears (Figure 1-34) and lists all replacement parts for the 74LS74A. Select a faster part, e.g. 74F74.
12. Click on the **STEP** button and notice that all timing violations related to the U2(74LS74A) device have now disappeared.

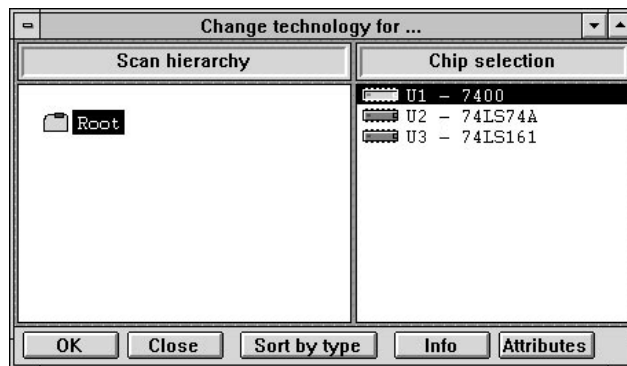


Figure 1-33. Change Technology Window.

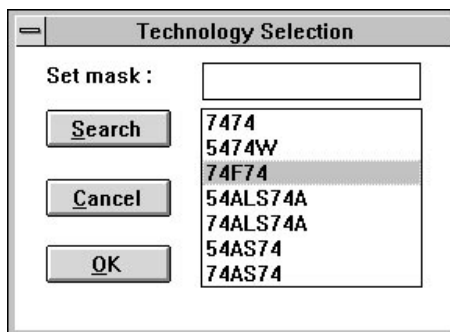


Figure 1-34. Device Replacement Window.

ACTIVE-CAD has many additional features that will aid you in design analysis. Most of these features are located in the **Options**, **Patching** and **Utilities** menus and in the toolbox. You can learn about them by reading the appropriate chapters in this *Guide*.

Unit Delay Simulation Mode

Unit delay simulation mode is a simulation mode with all propagation delays set to current simulation resolution. Its functionality is similar to the Glitch mode, but timings with more proportional time scale are produced.

Quick Application Notes

Deleting empty rows between signals

To delete empty rows, click in turn on:

- ☐ **Signal** menu
- ☐ **Delete** option
- ☐ **Empty Rows**

All empty rows are instantly deleted. The **Delete** menu is shown shown in Figure in Figure 1-35.

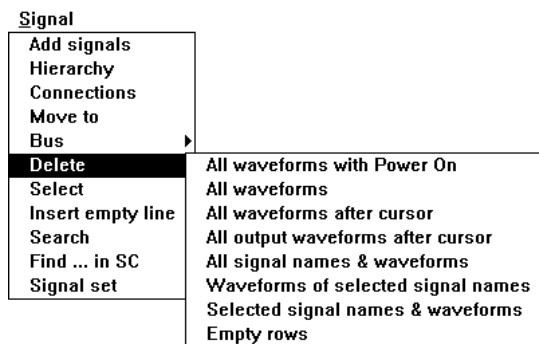


Figure 1-35. Signal Delete Menu.

How to work effectively with buses

In most cases, ACTIVE-CAD does not automatically create buses from the netlist information. Unless the netlist has been generated by ACTIVE-CAD schematic editor, you need to create these buses in the simulator editor. This editor gives you the freedom to define and redefine all buses as the simulation progresses.

Creating a bus

To create a bus, follow this procedure:

- ☐ Arrange all signal names in ascending or descending order, e.g. DATA0, DATA1, DATA2, etc. No other signal names can be present between these signal names.
- ☐ Click at the top-most bus signal name.
- ☐ Press the **[Shift]** key and click on the last member of the bus. Note that all signals between the first and last selected signal line turned blue.
- ☐ Click on the **BUS** option within the **Signal** Menu. When the window shown in Figure 1-35 appears, click on **Create**.
- ☐ In response, the simulator activates the **BUS** button. When this button is activated, all signals are converted into a single bus line.
- ☐ Toggle the **BUS** button and notice that first signal in the bus group is marked with an asterisk (*), indicating the name of the new bus. The other bus signals are marked with the plus (+) sign.

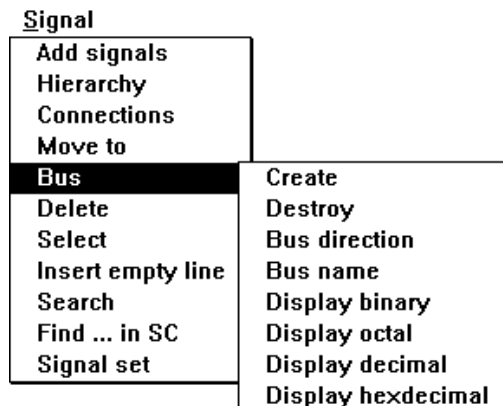


Figure 1-36. Bus Menu.

BUS display control

You can show a bus as a single bus line or as a composition of its discrete signal lines. To switch between these two modes of the bus display, click on the **BUS** button.

Selecting BUS display format

Buses can be displayed in binary, octal, decimal and hexadecimal formats. You can define and redefine the display format at any time during simulation. To select the appropriate bus format follow this procedure:

- ☐ Click on the selected bus
- ☐ Select from Figure 1-36 the appropriate bus format

Note that ACTIVE-CAD instantly converts the existing buses to the desired format.

Naming and renaming BUSES

The bus name is automatically assigned by ACTIVE-CAD when you create a bus. It is derived from the first signal name in the bus. However, you can change the bus name using the following procedure:

- ☐ Click on the selected bus name
- ☐ Select the **BUS Name** option from the window shown in Figure 1-36
- ☐ When the **BUS Name** window appears (Figure 1-37), enter the new bus name and click on the **OK** button

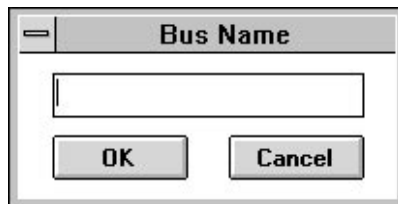


Figure 1-37. Bus Name Window.

Tracing design connectivity

The ACTIVE-CAD simulator allows you to trace the signal connectivity throughout all hierarchical levels. This is particularly useful if you don't have the latest schematics, or the designs are so big that it is difficult to work with the schematic sheets.

To trace device pin or signal name connectivity, follow these steps:

1. Click on the selected device pin or signal name, e.g. the *RST* signal shown in Figure in Figure 1-32.
2. When the selected item turns blue, click on the **Connections** option in the **Signal** menu.
3. Figure 1-38 displays the connections for the selected item.
4. Double-clicking on any of the pins listed shown in Figure in Figure 1-38 (node listing) will display all the pins of the selected device. Double-clicking on any of the device pins will display a new signal node associated with the selected pin.

This operation allows you to trace the signal distribution and verify the schematic-netlist compatibility.

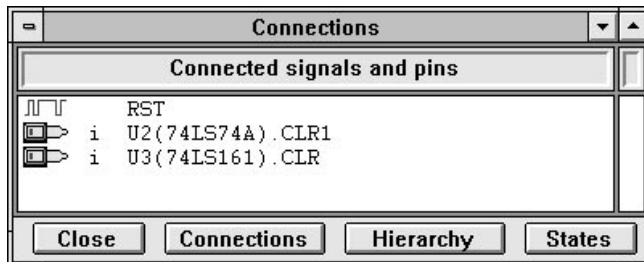


Figure 1-38. Signal Connectivity Table.

You may access the window shown in Figure 1-38 by clicking on the signal selection button in the Waveform Viewer toolbar, selecting a signal from Figure 1-8 and then clicking the RIGHT mouse button. ACTIVE-CAD will display the window shown in Figure 1-39. If you now click on the **View Connections** option, ACTIVE-CAD will display a window similar to Figure 1-38. The search for pin and signal line connections is identical as described above in reference to Figure 1-38.

<u>M</u> ove
M <u>o</u> ve <u>a</u> ll
<u>I</u> nsert empty line
<u>S</u> elect all
<u>D</u> eselect all
S <u>e</u> arch for ...
<u>B</u> us mode on/off
F <u>i</u> nd ... in <u>S</u> C
<u>V</u> iew Connections
<u>S</u> how Alias

Figure 1-39. Viewing Connectivity.

Searching for signal names, components and device pins

You can find signal names, devices and device pins in the ACTIVE-CAD simulator listings by clicking the RIGHT mouse button in any of the fields shown in Figure 1-8. In response, ACTIVE-CAD will display Figure 1-39. If you clicked in the **Pins For** field, clicking on the **Search for...** option will display Figure 1-40. If you clicked in the **Signal Selection** field, clicking on this option will display Figure 1-41. To search for pin and signal name locations do the following:

- ☐ Enter the signal or pin name shown in Figure 1-40 or 1-41
- ☐ Click on the **Select** button

ACTIVE-CAD instantly highlights the selected component or signal name in the listing. If you are using one of the schematic editors that have direct coupling with ACTIVE-CAD through DDE, DLL or OLE protocols, then the selected test points will also be marked on the schematic.



Figure 1-40. The Search for Pin Window.



Figure 1-41. Search for Signal Window.

Locating simulator data on schematic sheets

You can trace the location of devices, pins, signal names and macros on the schematic sheets directly from the ACTIVE-CAD simulator. Also any data selected in the schematic editor is displayed in the ACTIVE-CAD simulator. This cross-probing requires that the schematics have tight coupling with the ACTIVE-CAD simulator. SUSIE-CAD, ACTIVE-CAD, DIGILAB, and similar schematics have such interfaces.

Cross-probing between simulator and schematic designs greatly improves design comprehension and speeds its analysis. Ask the schematic editor vendors about their direct links with ACTIVE-CAD simulator.

Locating components on schematics

To locate any component on a schematic sheet, click on its designation in the **Chip Selection** window of Figure 1-8. Next, click the RIGHT mouse button over the **Chip Selection** window to display a local menu. Select from that local menu the **Find ... in SC** option. It instantly switches the display to the appropriate schematic sheet and shows the selected component location with a red box around the component

Locating device pins on schematics

To locate any component on a schematic sheet, click on its designation in the **Pins For** window of Figure 1-8. Next, click the RIGHT mouse button over the **Pins For** window to display a local menu (Figure 1-39). Select from that local menu the **Find ... in SC** option. It instantly switches the display to the appropriate schematic sheet and shows the selected component pin location with a red blob.

Locating signals on schematics

To locate any signal on a schematic sheet, click on its designation in the **Signals Selection** window of Figure 1-8. Next, click the RIGHT mouse button over the **Signals Selection** window to display a local menu, which is similar to Figure 1-39. Select from that local menu the **Find ... in SC** option. It instantly switches the display to the appropriate schematic sheet and shows the selected signal by turning it red.

Applying signal waveforms at any screen location

Most simulators require you to apply test vectors at the time t_0 or at the beginning of the simulation. However, since ACTIVE-CAD is a real-time interactive simulator, you can apply your signals at any time and at any screen location during the simulation. The process of using formula-based signal waveforms is comprised of two steps:

- ☐ Creating or editing formula-based signal waveforms
- ☐ Assigning signal waveforms to the selected screen locations

Creating formula-based signal waveforms

To access the formula editor, select in sequence the **Waveform** menu and then its **Formula** option. In response, ACTIVE-CAD displays shown in Figure 1-42 the **Formula** options submenu.

NOTE: the Formula Editor described in reference to Figure 1-20 generates signals that start at time t_0 . The waveform formula described here applies the formulas to the signal waveforms at the desired time t_n .

If you select the **Edit** option, ACTIVE-CAD displays Figure 1-43 which allows you to enter a signal waveform formula. This editor works similarly to the one described in **Formula Stimulators**, except:

- ☐ It does not assign any names to the generated signal waveforms so they cannot be automatically assigned to any signal names. Instead, they have to be manually inserted into the on-screen waveforms.

Since these waveforms can be fed at any screen location, they are the basic tool for dynamic what-if analysis, in which segments of waveforms are modified and resimulated to test design modification concepts.

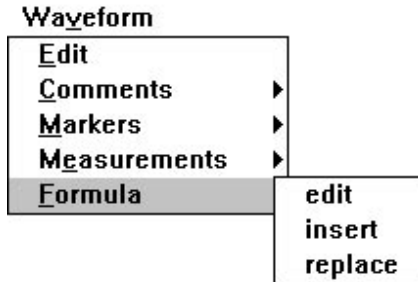


Figure 1-42. Formula Menu.

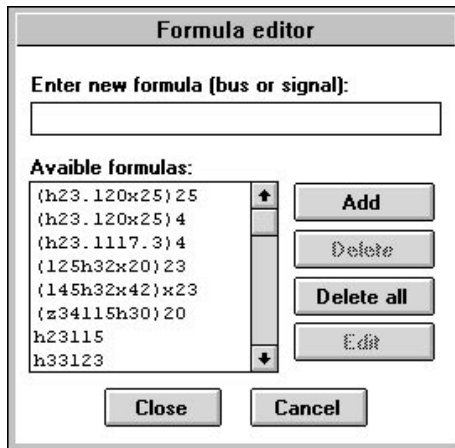


Figure 1-43. Formula Signal Editor.

The editor shown in Figure 1-43 allows quick and efficient development of signal waveforms of any shape and duration. They can be used for non-repetitive functions and bursts of pulses of any duration with various logical states.

The formula stimulators give you total freedom to define and redefine signal waveforms while the simulation is in progress. The process of defining a formula is comprised of:

- ☐ parentheses (used in nested expressions)
- ☐ logical state symbols: H,L,Z,X
- ☐ hexadecimal bus values; e.g. [1FA5], [3C8], etc.

- ☐ durations of the signal states (e.g. 12.31 nanoseconds)
- ☐ number of repetitions of the signals listed in parentheses

Example 8:

Type into the **Enter New Formula (Bus or Signal)** field shown in Figure 1-43:

(H2L3)4; High for 2 ns, Low for 3 ns, repeat 4 times

Next, click on the **Add** button. The formula is automatically added to the **Available Formulas** window and will be available for editing signal waveforms. The above formula will result in the following waveform:

This segment can be added at any screen location and as many times as needed.

NOTE: There is no limit on the nesting of signals and on their duration.

Example 9:

Enter into the **Enter Formula (Bus or Signal)** field:

([3FC5]25.2[053A]33.4)25

This formula will generate a 16-bit bus signal. The first segment of the bus has a hexadecimal value of 3FC5 and lasts for 25.2 nanoseconds. The second segment has a hexadecimal value of 053A and lasts for 33.4 nanoseconds. Both bus segments will be repeated 25 times.

There is no limit on the nesting depth of bus statements nor on duration of bus signal segments. You must, however, always include the [] brackets and list the duration of each bus segment. All bus segments in the formula must have the same number of hexadecimal characters (the same number of bus lines). For example, you cannot mix 12- and 16-bit buses, e.g. formula ([3FC5]25.2[53A]33.4)2 is incorrect because the bus widths in the [] parenthesis are different.

You can create and save tens of formulas for use with ACTIVE-CAD. If they overfill the **Available Formulas** window shown in Figure 1-43, the window will change and you will be able to scroll its contents with the arrow buttons.

Applying formula-based signal waveforms

To apply a formula-based signal waveform:

First, click at the screen location where you want to apply the selected signal waveform. A blue cursor will appear at that location and the associated green cursor will highlight the selected signal name.

Next, select either the **Replace** or **Insert** option from the window shown in Figure 1-42. The **Replace** option will allow you to override the existing signal waveforms with the new ones. The **Insert** option inserts the applied signal waveform at the selected screen location and shifts the existing waveforms to the right.

Click on the selected signal formula in the **Available Formulas** window shown in Figures 1-42 and then click on the **OK** button. The selected signal waveform will be instantly placed at the selected screen location.

The formulas created in Figure 1-20 and those created in Figure 1-43 are different. The formula editor in Figure 1-20 assigns concrete names to each formula, forming independent signal waveform entities. These formulas can be assigned by clicking directly on the signal names.

Since the formulas from Figure 1-20 always calculate the logical states from the time t_0 or beginning of simulation, independent of when they are used in the simulation process, they always produce the same logical states at some future time t_n .

The formulas in Figure 1-43 have no names, and can thus be fed only at concrete screen locations. Also, they cannot be edited thereafter because they are automatically integrated with the existing signal waveforms, or as one would say, they disappear in the crowd of other waveforms.

Simulation precision

All timing simulators operate with a certain precision. This precision or resolution is fixed for most simulators and is typically either 100 picoseconds or 1 nanosecond. The simulation time is generally limited to 100 milliseconds at 100 picoseconds resolution and one second at one nanosecond resolution. One hundred milliseconds in the life of an ECL device is a long time. The ECL device may process an immense amount of data. However, an industrial controller may complete in that time only a few operations. So if you are working with devices that operate with microsecond speeds, not much can be simulated with simulators that have such high resolution.

To accommodate a broad range of electronic devices and circuits with vastly different timing characteristics, ACTIVE-CAD has a variable simulation precision ranging from 10 picoseconds to 1 millisecond. This means that you can simulate from 40 milliseconds to over one hour of device operation time. This broad time range allows you to simulate both the high speed gallium arsenide circuits and slow industrial controllers.

To set a new simulation precision, select in sequence:

1. **Options** menu (Figure 1-45)
2. **Simulation Precision** option
3. The desired simulation precision from the **Select Precision of Simulation** window shown in Figure in Figure 1-44.

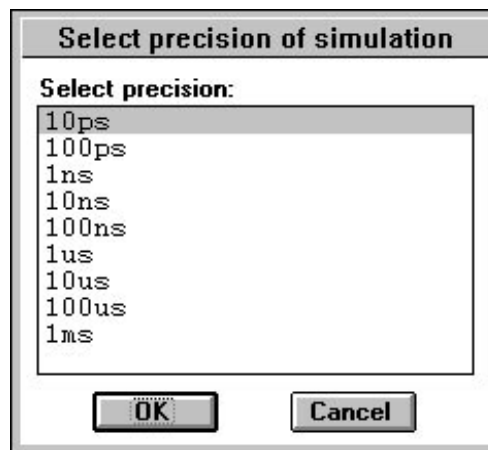


Figure 1-44. Selecting Simulation Precision.

Upon rescaling the simulation precision, ACTIVE-CAD moves the cursor to the time t_0 so you cannot mix (in the same simulation run) timing waveforms with different resolutions.

Simulation time estimate

Sometimes simulation may take longer than expected. If you want to be warned beforehand about the long simulations, select the **Options** menu (Figure 1-45), and then click on the **End of Step Estimation** option.

Each time the simulation step takes longer than 10 seconds, ACTIVE-CAD will display a window with the time needed to complete the current simulation run. This window, shown shown in Figure in Figure

1-46, counts down the simulation time so that you know how much longer the simulation will take.

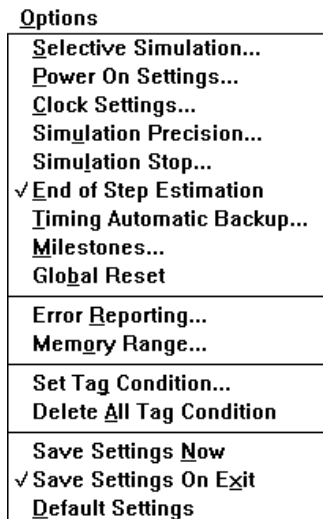


Figure 1-45. Options Menu.

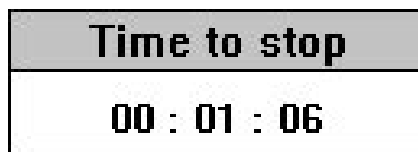


Figure 1-46. Time to Complete Simulation Window.

Automatic backup of simulation results

To protect yourself from losing valuable simulation data, use the automatic data backup option by selecting the **Timing Automatic Backup** option from the **Options** menu shown in Figure 1-45. In response, ACTIVE-CAD displays the screen shown in Figure 1-47 which allows you to enter the time intervals at which an automatic backup should take place.

If you are working in an interactive mode, set that backup to between 5 and 10 minutes. If you are working in a batch mode or with long test vectors, set the backup to half hour intervals.

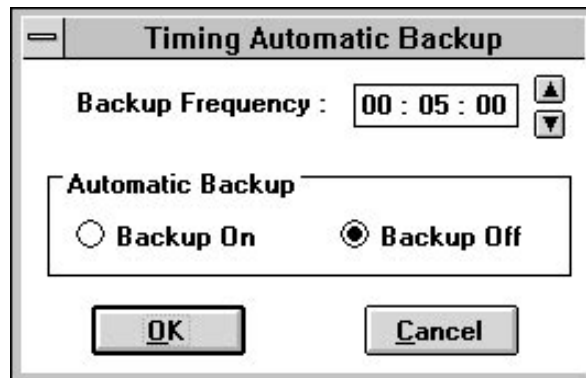


Figure 1-47. Timing Backup Option.

Global design reset

The **Global Reset** option in the **Options** menu allows you to reset all devices to their initial state without returning to the time t_0 . This means that you can reset the design at any time t_n and continue simulation, starting with the test vectors that existed at the time t_n .

Design Error Handling

Error reporting

You can dynamically select which design errors should be reported or stored, eliminating errors that obscure your current analysis. To set up the error reporting procedure, select the **Error Reporting...** option from Figure 1-45. In response, ACTIVE-CAD displays the table shown in Figure 1-48. Toggle the Xs in the boxes to indicate which errors you want reported. You must set the **Work Mode** to **ON**. Otherwise, all error reports will be disabled. However, the red error markings on the signal waveforms will continue to appear irrespective of these settings.

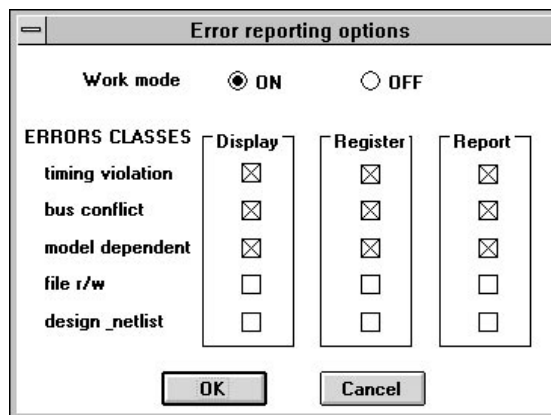


Figure 1-48. Error Reporting Window.

Error Classes:

- ☐ **Timing Violation;** reports any timing violations related to setup and hold times, clock widths, pulse widths, relationships between clock and reset or preset, etc. You can only select or deselect all of the errors. You cannot select only some types of timing violations.
- ☐ **BUS conflict;** reports all violations involving conflicting logical states in signal nodes
- ☐ **Model Dependent;** each model has embedded error messages. If certain conditions for proper model operation are not met, e.g. setup time, the appropriate model-embedded message will be displayed.
- ☐ **File R/W;** monitors and reports hard disk Read/Write errors
- ☐ **Design Netlist;** lists errors detected during netlist import

Error handling

- ☐ **Display;** displays a detailed error message window. The red error markings on the signal waveforms are always present, independent of whether you have selected this feature.
- ☐ **Register;** saves the selected errors into a Message Box for future reference
- ☐ **Report;** enters the selected errors into the simulation error report which is enabled from the **Error Viewer** option of the **Utilities** menu

Correcting design timing errors

A design can have timing errors related to device input signal violations (e.g. setup and hold times) and ones related to layout delays. You could in some cases substitute a faster part to correct both device timing viola-

tions and layout delay problems. However, as a general rule for high performance circuit designs, you need to analyze these errors separately and then make the decision on how the problem should be handled.

If you have found a timing violation that is related to one of the devices, you can resolve the problem by analyzing:

- ☐ Propagation paths on the failing device inputs
- ☐ Layout delays
- ☐ Device timing characteristics

Since ACTIVE-CAD timing violation reports indicate how severe the problem is, you may at the very outset determine if the problem is curable by the device (technology) replacement, an architectural design change or layout modifications.

Checking for the worst-case test condition

There is no reason to correct a design that has not been tested over the full range of loading, temperature, voltage, device propagation and other circuit constraints. You need to analyze the data paths and set them to the worst case conditions. For example, if you are testing U5A for the worst-case setup time condition (Figure 1-49), you should set the U1A and U2A devices to their maximum propagation delays and the U3A and U4A devices to their minimum propagation delays. Similarly, if you are testing U5A to the worst-case hold time conditions, you should set U1A and U2A to the minimum propagation delays and U3A and U4A to the maximum propagation delays.

If you use gates located on the same silicon device for both D and CLK channels, their minimum and maximum timing parameters will be identical. These gates will equally effect both signal channels and their actual value will have no effect on the setup and hold times. Since U1A and U3A are 7400 type gates and since 7400 has four such gates in a single package, you should use the U1 gates for both D and CLK signal channels to minimize the effect of device timing parameters.

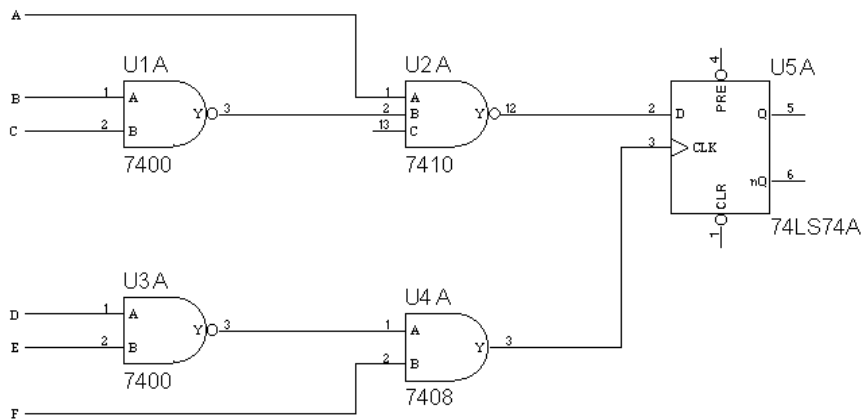


Figure 1-49. TEST_D - Worst-Case Timing Analysis.

Example 10:

This is an analysis of the worst-case U5A setup time. If you have used for U1A and U3B the same device package (e.g. U1A and U1B), you do not need to analyze the propagation delay effects of these gates. Any propagation delay value will produce the same effect on both the *CLK* and *D* input signal paths, thus compensating for each other. To set U2A (74LS10) to the maximum propagation delays, select the **Editing Timing Specification...** from the **Patching** menu. When Figure 1-50 appears, double-click on the U2 device. In response, ACTIVE-CAD displays the U2 gate timing parameters in Figure 1-51. Click on the **Max** button and note that the delay values in the **Set** column, which are used by the ACTIVE-CAD simulator, are set to the maximum propagation delays. Click on the **OK** button to exit this window.

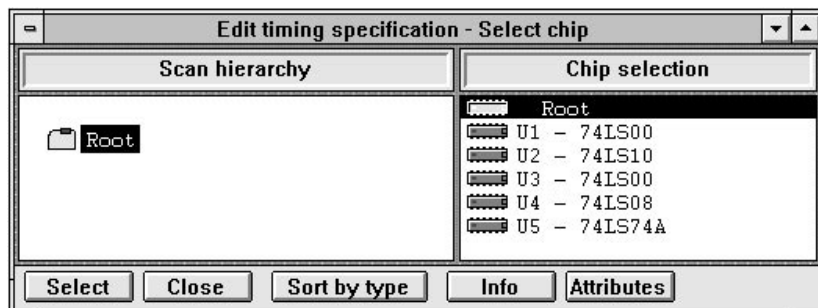


Figure 1-50. Selecting a Device for Editing.

Name	Min	Avg	Max	Set	Description
Tp1LH	5.2ns	9ns	15ns	9ns	input to Y L->H delay
Tp1HL	5.2ns	10ns	15ns	10ns	input to Y H->L delay
Tp2LH	5.2ns	9ns	15ns	9ns	input to Y L->H delay
Tp2HL	5.2ns	10ns	15ns	10ns	input to Y H->L delay
Tp3LH	5.2ns	9ns	15ns	9ns	input to Y L->H delay
Tp3HL	5.2ns	10ns	15ns	10ns	input to Y H->L delay

OK Cancel % of Max

Figure 1-51. Timing Parameters for 74LS10.

Since the worst-case setup time requires setting U4 to minimum propagation delay values, double-click on the U4 device in Figure 1-50, and then click on the **Min** button shown in Figure 1-51. Return back to the ACTIVE-CAD simulator and simulate the design. If the design shown in Figure 1-49 does not show any hold time violations, no hold time violations will appear either.

Each setup of propagation delays which has caused the design failure should be saved for future reference. You need to accumulate and map the design problems before you take any action.

Analyzing ASICs for setup and hold times

If you are analyzing an ASIC design, you do not have to set up each cell to its minimum or maximum value because the cells behave just like the gates U1A and U3A shown in Figure 1-49. They track each others propagation delays automatically and their minimum or maximum propagation values will have the same overall effect on the circuit behavior. The only test you need to perform is to set the entire design to the minimum and maximum propagation delay values. To set up the entire design or any design macro to the specific propagation delays, follow up the below procedure for global propagation delay setups.

Global design propagation delays setups

You can set the entire design or any of its sections to the selected propagation delay. This is very useful when working with ASICs because with a click of the mouse button, the entire design is rescaled to the desired propagation.

To change design delays in a global manner, double-click on the ROOT or macro in the **Scan Hierarchy** field of the window shown in Figure 1-50. In response, ACTIVE-CAD displays Figure 1-52 which allows you to set the propagation delay of the selected design section to:

- ☐ Minimum
- ☐ Maximum
- ☐ Average
- ☐ % of the maximum value

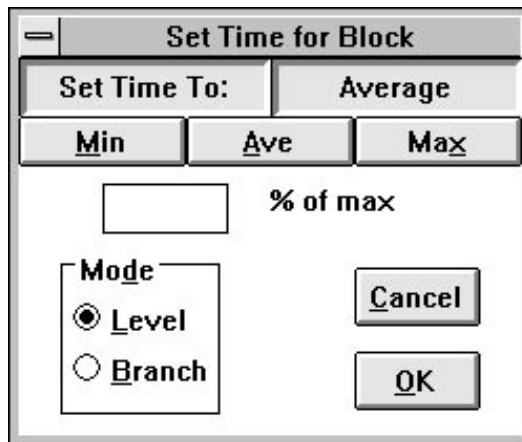


Figure 1-52. Scan Hierarchy for Global

The % of Max value is used to rescale the propagation delays due to temperature and voltage. The loading effects are calculated by ACTIVE-CAD or they are provided by the ASIC manufacturers through the post-layout netlists of the devices.

If you select the **Level** option shown in Figure 1-52, only cells and devices at that hierarchical level will be affected. The macros at that level will not be affected and they will retain their original settings. Selecting the **Branch** option will effect all the components, including the macros at the selected level.

Device related timing violations

If you want to check if a faster part would cure the problem, select **Change Technology** from the **Patching** menu. In response, ACTIVE-CAD displays Figure 1-53. Double-click on the selected device in the **Chip Selection** field. When the window in Figure 1-54 appears, select the appropriate part and click on the **OK** button. The part is instantly replaced in the simulator executable tables and fed back to the schematic editor if it is directly connected through DDE, DLL or OLE protocols.

If you continue simulation, you will notice that the new device has taken control of the timing display and shows a different timing performance.

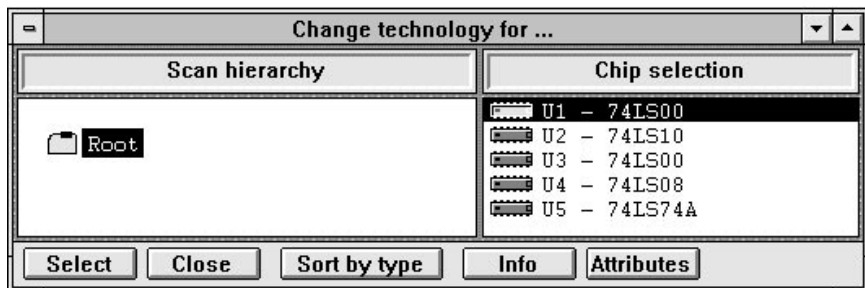


Figure 1-53. Selecting part for replacement.

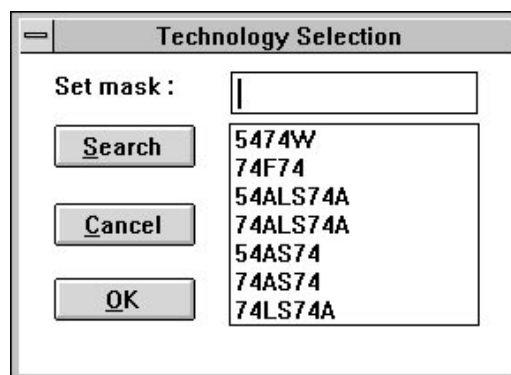


Figure 1-54. Replacing a part.

Analyzing line delays

Line delays have a considerable effect on high speed designs. To analyze their effect, select the **Change Line Delays** option from the **Patching** menu. In response, ACTIVE-CAD displays the **Scan Hierarchy** window. Double-click on the selected hierarchy level for line (layout) delays. If there are any delays, ACTIVE-CAD will display a table with delay values. You can edit these values and emulate new layouts.

For example, analyzing the actual layout and proposed changes, you can estimate the new propagation delay. To verify any possible side effects of these changes, you can enter the expected new line delay values in the line delays table and simulate the changes. Only if ACTIVE-CAD confirms the proper design operation should you implement the proposed changes.

Simulating Very Large Designs

ACTIVE-CAD does not care how big your design is. It can load and simulate practically any design size if your computer has enough RAM. For example, some users have reported that they have simulated designs in excess of 300,000 gates using just 32 Mbytes of RAM. However, for efficient simulation, you need at least 8 Mbytes of RAM for the basic software (Windows, ACTIVE-CAD simulator, libraries of models and symbols, etc.), plus 1 Mbyte of RAM for each 6,000 to 10,000 gates in your design. The exact amount of required RAM depends upon the design topology.

All commercial simulators simulate the entire netlist, which causes large designs to simulate very slowly. If the design is big enough, it may slow the simulation process to a point where it becomes impractical to use it effectively.

The simulation speed is the main constraint on the simulated design size. To speed the simulation process, hardware accelerators were touted in the late 80s. However, while they had some application in ASIC designs, they completely failed at the system level where some device models were missing or were not available in the format required by the hardware accelerators.

A new method of dealing with large designs is selective simulation and incremental design process (*read* An Introduction To Simulation And Virtual Hardware, issued by ALDEC, Inc.). The selective simulation process allows you to simulate only the desired design sections, down to the lowest hierarchical and component level. You can select and deselect any design section while the simulation is in progress and by decreasing the size of the simulated section, you can increase the simulation speed. Since the simulation speed is proportional to the size of the simulated circuit, you can effectively simulate the selected design sections independently of how large the overall design is.

The patented selective simulation process is available with SUSIE 6.0 and ACTIVE-CAD products only.

Selective design simulation option

Load the netlist *TEST_H* (provided with your software), and click on the **Selective Simulation** option in the **Options** menu. In response, ACTIVE-CAD displays the window shown in Figure 1-55. The **Scan Hierarchy** window shows the hierarchical design structure, and the **Chip Selection** window lists devices and macros at the selected hierarchical level.

Clicking at any hierarchical level in the **Scan Hierarchy** window displays in the **Chip Selection** window all the macros and devices at that hierarchical level. By clicking on these devices and macros you can enable and disable them from the simulation. Disabling a device sets its output pins to the high-impedance logical state. You can override these pins with the desired signal waveforms to emulate the operation of the disabled design sections.

All enabled chips and macros are dark. All disabled ones are white. The selections are instantly affecting the simulation process.

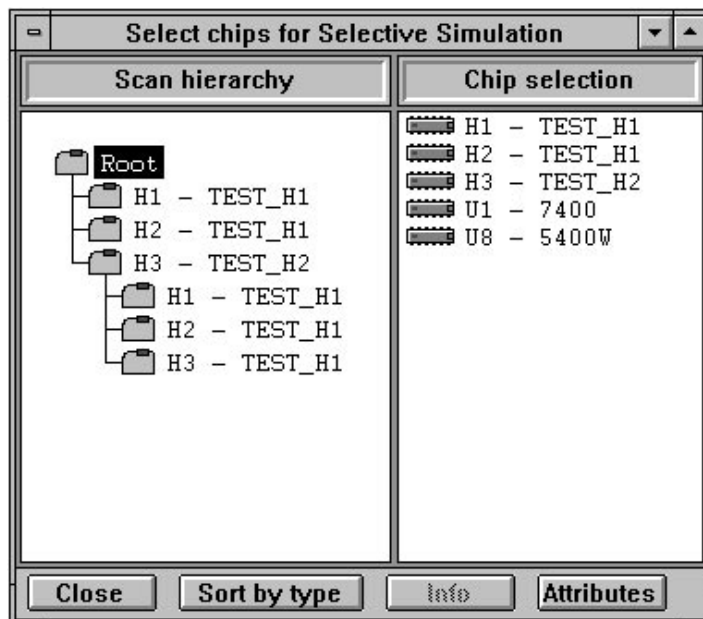


Figure 1-55. Selective Simulation window.

Example 11:

Load the Test_H netlist and copy to the simulator's **Signal** field the input and output pins of gate U1A, located at the top hierarchical level. Assign to these input pins the stimulators shown in Figure 1-56. Simulate one **LONG** step. Next, select the **Selective Simulation** from the **Options** menu and when the window in Figure 1-55 appears, click on the U1 device in the **Chip Selection** field to disable it. When the device body shown in Figure 1-55 turns white, indicating a disabled device, close the window and simulate another **LONG** step. Note that the U1.Y1-3 output is now floating. After both **LONG** steps your display should look like the one shown in Figure 1-57.

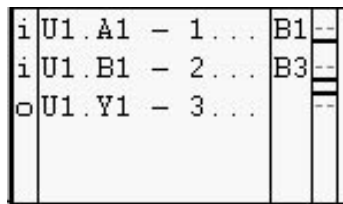


Figure 1-56. Simulation of gate U1A.

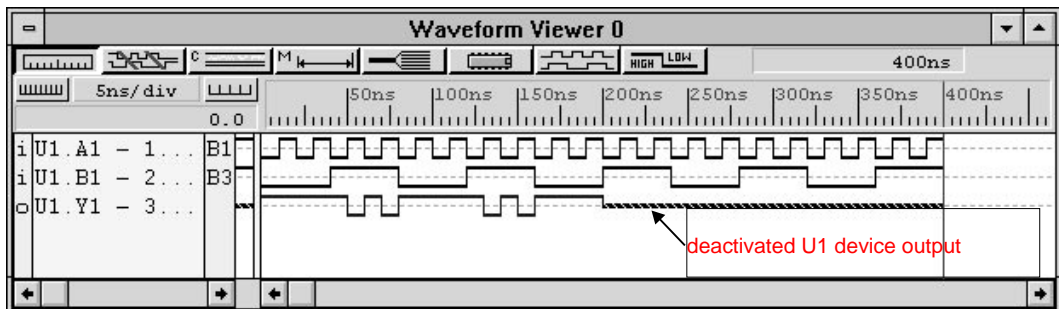


Figure 1-57. Activated and Deactivated Gate Simulation

As an additional exercise, you can select the U1A gate from the lowest hierarchical level in the *TEST_H* netlist (gate U1A within the H1 macro which is located within the H6 macro), and then simulate this gate by:

- ☐ enabling and disabling the U1A gate directly
- ☐ enabling and disabling the macro H1 that includes the U1A gate
- ☐ enabling and disabling the macro H6 that includes the above mentioned macro H1

Note that when you enable and disable the macros H1 and H2, you get the same effect as when you directly enable and disable the U1A gate.

The selective simulation option allows you to select and simulate any combination of macros and devices from any hierarchical level.

Resetting A Design

Resetting or presetting a design to a desired logical state is the most complex operation in the simulation process because it involves special features in both the simulator and IC models.

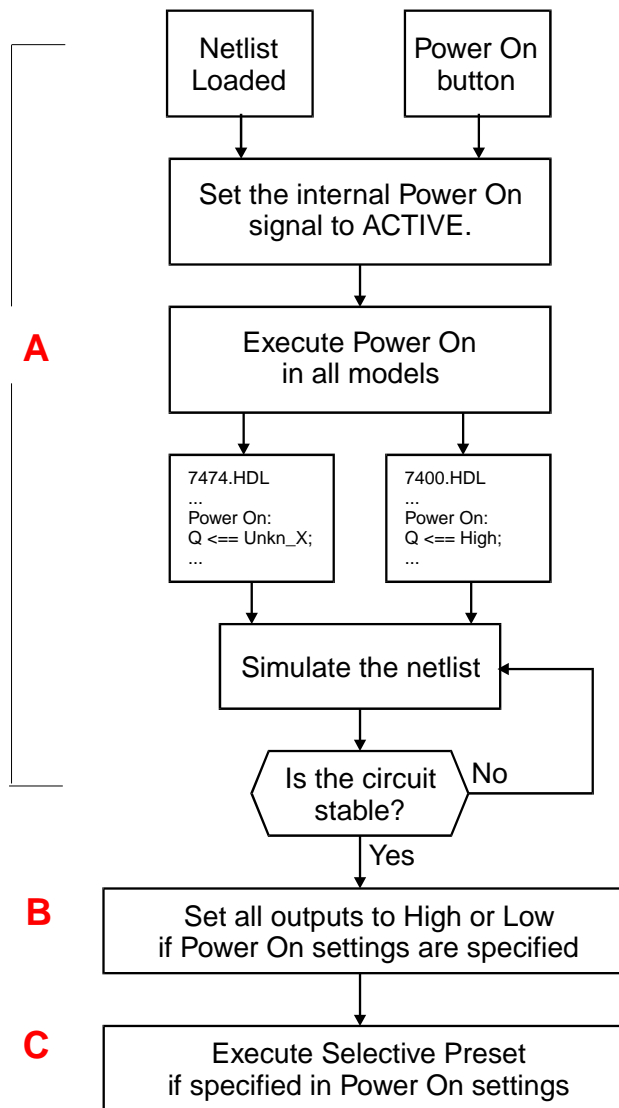


Figure 1-58. Power On Process Flowchart.

The design resetting process is shown in Figure 1-58. It is comprised of:

- ☐ (A) - Executing the **Power-on** instructions in all models and simulating the design
- ☐ (B) - Asserting the Power-on Settings on device outputs
- ☐ (C) - Executing the **Preset** operation, if specified in the Power-on Settings

Power-On model instructions

Activating the **Power-on** button in the simulation toolbox forces all models to execute their internal power-on instructions. Some models like combinatorial gates, decoders, etc., do not have Power On or Global Reset instructions. Typically, all devices are set to the X-Unknown signal state, except for Altera parts which are set to logical zero. After the IC models are preset to their power-on states, ACTIVE-CAD simulates the design till a stable state is achieved. If there are more than 10,000 oscillations, ACTIVE-CAD declares an unstable design condition and displays an error message.

Power-on Settings

After the design has achieved its steady-state, ACTIVE-CAD executes the Power-on Settings. These settings are factory-provided as default Power-on parameters. You can change them to fit your requirements, however, the new parameters will affect only the current simulation. Each time you start ACTIVE-CAD, the power-on parameters will automatically revert to the default factory settings.

You may change or view the power on setting in the **Power_On Setting** window (Figure 1-59) which is displayed by clicking on the **Power On Settings** option within the **Options** menu.

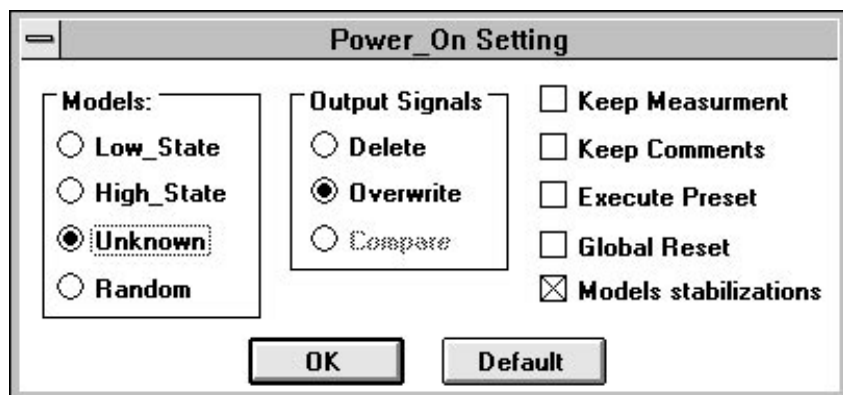


Figure 1-59. Power On Settings Window.

The default power-on settings are as follows:

- ☐ models are set to the unknown (X) state
- ☐ newly generated outputs override the old ones

- ☐ old comments and measurements are deleted
- ☐ **Execute Preset** is not activated

By selecting the appropriate options shown in Figure in Figure 1-59 you can change the above settings. You must, however, remember that the logical levels applied by the power-on settings to the device pins remain on these pins till they are overridden by the first signal transitions generated for these pins by the simulator. This may at times produce visually wrong logical levels at gate outputs that did not have any post-power-on signal transitions.

Preset

If the **Execute Preset** option in the **Power_On Setting** window (Figure 1-59) has been selected, then it will be executed immediately after the power-on settings have been completed. However, you must select the preset conditions (see *Presetting a design to the desired state* section in this Chapter) or load the preset file before clicking on the **Power-on** button.

If no preset conditions have been set in ACTIVE-CAD, then no preset conditions will be executed during the power-on procedure.

NOTE: Since the **Power-on**, **Power-on Settings** and **Preset** may set an output to conflicting states, you need to carefully review each effect to achieve the desired power on operation. To simplify the design analysis use the **Power-on Settings** and **Preset** very sparingly.

Global Reset

ACTIVE-CAD allows you to have two resets for each IC model, one for power-on and one for setting a special logical level on the device output(s). For example, the Power-on Reset may set output pins to the unknown (X) state and the Global Reset may set these outputs to a Low logical level.

The Global Reset is executed by selecting the **Global Reset** option within the **Options** menu. All models in which this reset instruction is implemented within the source code will respond to this operation. Models without Global Reset instructions will ignore this command.

NOTE: Most of the ACTIVE-CAD models, except for some FPGA libraries (Actel, Xilinx) do not currently have the Global Reset implemented.

Setting simulation reference points (Milestones)

To save RAM memory, simulators save simulation data only for those test points that are displayed on the screen. The logical states of all other test points which are not listed in the simulator Waveform Viewer are stored in the RAM memory only for the duration of the current simulation cycle and are overridden with the next cycles data. To compensate for this complete loss of data, ACTIVE-CAD has a milestones option which allows you to return to a selected simulation cycle and set the entire design to the exact design conditions that existed at that time. This option has several major advantages over batch-mode simulators:

- ☐ Rather than being forced to start the simulation from time t0, you can restart the simulation from any past simulation cycle
- ☐ ACTIVE-CAD allows you to add new test points to the screen, starting from any past simulation cycle
- ☐ You can change the design at any past simulation cycle, which is marked by a Milestone, and view the improvements
- ☐ ACTIVE-CAD lets you change test vectors at any past simulation cycle to force new system behavior
- ☐ You can resimulate the design from a cycle just prior to an error

The milestones option saves you time because you are no longer forced to resimulate the design from the time t0. Instead, you can use any marked simulation cycle (Milestone) as the starting point.

There are three mechanisms for creating the milestones:

- ☐ automatic (periodical)
- ☐ manual
- ☐ breakpoint-driven

Automatic Milestones

To put some milestones along the simulation highway, click on the **Milestones** option within the **Options** menu. In response, ACTIVE-CAD displays the **Milestones** window shown in Figure 1-60 which allows you to select the number of milestones to be placed on the screen and the time interval between them. Enter the separation between milestones into the **Period** window. If you do not specify the time unit, it will be in nanoseconds by default.

Since the automatic milestones can fill the entire hard disk in less than an hour, ACTIVE-CAD has a built in self protection mechanism which asks

you how many milestones should be saved. If you specify eight (8), then a maximum of eight milestones will be saved and the 9th milestone will overwrite the first one. This provides for a shifting window of milestones that tracks the simulation process and allows you to return to any selected milestone within that area.

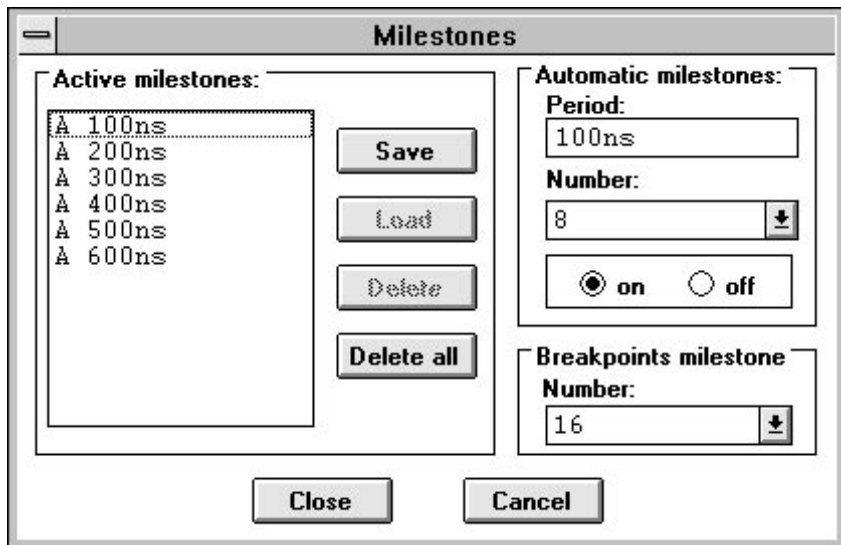


Figure 1-60. Milestones Selection Window.

After you have selected the number of milestones to be saved and their time interval, you need to activate the milestones by setting the **On** option (Figure 1-60). Otherwise, the milestones option will be inactive.

Manual Milestones

For greater convenience, ACTIVE-CAD allows you to manually set any simulation cycle (design condition) as a milestone. If you click on the **Save** button shown in Figure 1-60, ACTIVE-CAD will save the current simulation status as the reference point and will always be able to return to it during the simulation process. ACTIVE-CAD allows you to have an unlimited number of such manual milestones.

To return the simulator to any of its past simulation cycles, as represented by the milestones, select the **Milestones** window shown in Figure 1-60, click on the desired milestone in the **Active Milestones** field and then activate the **Load** button. Confirm your selection by clicking on the **Close** button. ACTIVE-CAD instantly sets the entire design and test vectors to the appropriate logical states and then positions the cursor at the desired (milestone) screen location.

Figure 1-61 shows a waveform diagram of the device pin U1.A1-1 with two additional simulation reruns (for the same pin), both starting at the M2 milestone. These reruns show different waveform patterns to the right of the M2 milestone line, which may be caused by applying new test vectors or performing design changes. This simulation flexibility from any past cycle is a unique feature of the real-time simulators such as ACTIVE-CAD, and allows direct tweaking of design behavior in real time.

Breakpoint-driven Milestones

You can set a milestone on a breakpoint condition. Such milestones are described in the *Breakpoints* chapter.

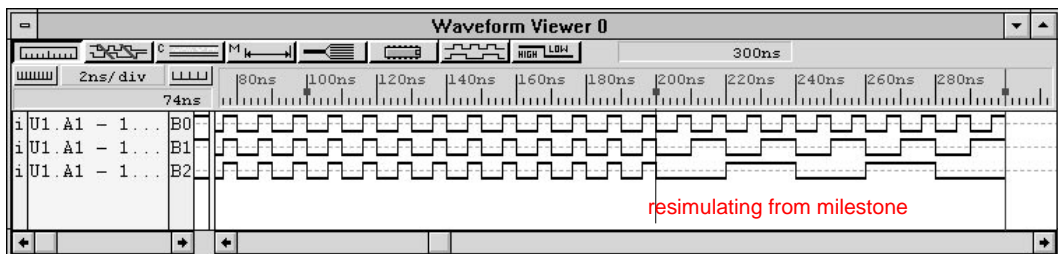


Figure 1-61. Multiple Simulations from a Selected Cycle

Milestones allow you to quickly verify various design concepts in an incremental fashion, without lengthy setups and compilations. Effective use of milestones is the cornerstone of real-time interactive simulation and to help your productivity, ALDEC has included this feature even in the lowest cost software versions.

Running long simulations

If you need to run a long design simulation, e.g. for five hours, you should click on the **Simulation Stop** option within the **Options** menu. In response, ACTIVE-CAD will display the **Start Long Simulation** window (Figure 1-62). Enter into the **Simulation Running Time** field for how long, in hours, minutes and seconds, you will run the simulation. You can enter this data directly or you can place the cursor in the appropriate field and use the up-down arrows to set the desired simulation time values.

To activate the long simulations timer, select the **Sim. Till End** option and then click on the **Start** button. It is recommended that the **Stop But-**


ton be activated as well, so that you can interrupt simulation when needed. The only time you would probably want to disable the **Stop** Button is when you are leaving the office and do not want anybody to interfere with the simulation process.



Figure 1-62. Long Simulation Setup Window.

Overcoming the 4,000 test vectors limit

The test vector is a vertical time slice of the waveform screen display with some signal transitions. Each test vector must have at least one signal transition. However, typical test vectors will have multiple simultaneous signal transitions, also called events. SUSIE-CAD products are limited to a maximum of 4,000 test vectors. You can, however, simulate any number of test vectors if you follow this simple procedure:

- ☐ If you have reached the 2,000 test vector limit, save the existing simulation results using the **Save Test Vectors** option in the **File** menu
- ☐ Click on the  **Waveform Delete** button to delete all waveforms from the screen
- ☐ Reload the test vectors; if you are using the binary counter, keyboard keys or formulas for test vector generation, you do not need to do any setups
- ☐ Since deleting signal waveforms leaves the design in logical states generated by the last simulation cycle, you can continue simulation by activating the **Step** and **Long (Step)** buttons

NOTE: When simulating design in test vector segments, do not activate the **Power-on** button, because it will delete the last simulation cycle data.

How to simulate large memory devices

To fully simulate large memory chips, you would need to use very expensive workstations, with a lot of RAM memory. Fortunately, most of the designs can be verified by simulating only the first few or the last few Kbytes of RAM code. And this can be done on any PC.

To specify how much memory should be simulated, select the **Memory Range** option from the **Options** menu. When ACTIVE-CAD displays the setup window shown in Figure 1-63, enter into the **Lower Mem Address** and **Upper Mem Address** the desired address space (number of bytes). You can select any value from the pull down list or type any specific address range.

The default address space is 1024 bytes, both for the lower and upper RAM limit. If you want to simulate other memory ranges, you need to enter the new values before starting the simulation process.

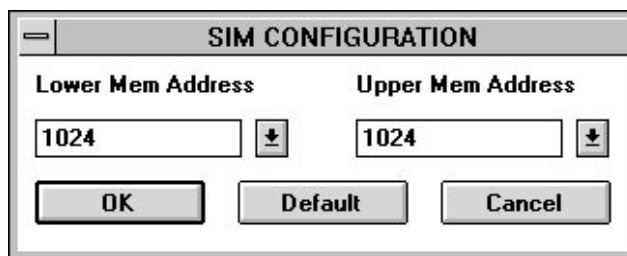


Figure 1-63. Selecting Memory Range for Simulation.

NOTE: The memory range that you set in the **Memory Range** option is applied to all memory devices in the entire design, including the internal microcontroller ROM and RAM.

How to search for selected signal conditions

At times you may need to find a certain set of signal conditions in the design. ACTIVE-CAD comes with the **Tag** option which facilitates such a search both forwards and backwards. The **Tag** can be set at any time during simulation and it will instantly display the Tag conditions in the past and new test vectors (signal waveforms).

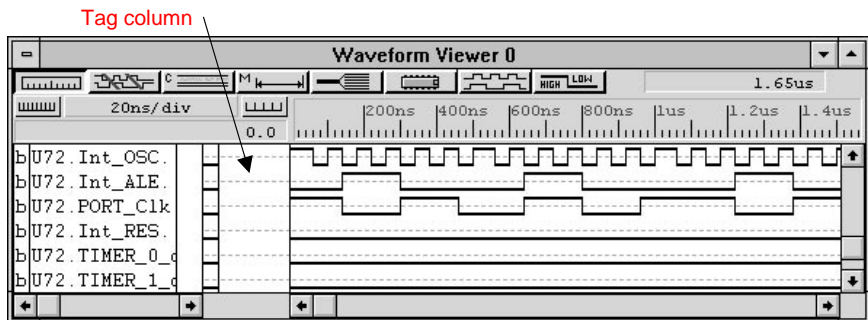


Figure 1-64. Tag Column Location

To select the **Tag** option, click on the **View** option in the **Utilities** menu. When the **View** menu appears, click on the **Tag** option. In response, ACTIVE-CAD displays a new column, shown shown in Figure 1-64. This column is used for entering the Tag signal conditions. ACTIVE-CAD searches for the logical AND of all listed signal conditions.

Creating Tag conditions

To enter the signal conditions you are looking for, select the **Set Tag Condition** option from the **Options** menu. In response, ACTIVE-CAD displays a table of all available signal logical states (Figure 1-65). Click on the signal name in the **Signal** field of Figure 1-64 and when it turns blue, click on the desired logical state in the table of Figure 1-65. Note that ACTIVE-CAD has entered the selected signal state in the **Tag** column. Select some additional signal conditions you want to include in the **Tag** search operation.

At least one of the **Tag** signals must be in a transition(e.g. LOW to HIGH) ACTIVE-CAD will search for the AND condition of all **Tag** signal states.

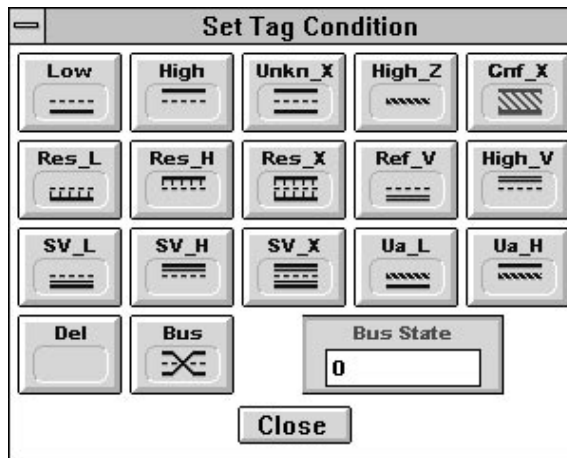


Figure 1-65. Logical States Table for Tag

The **Tag** option allows you to search for any signal transitions. To set up a signal transition, select the signal name in the **Signal** field. When it turns blue, click on the first signal logical level in the table of Figure 1-65. Next, click on the second signal logical state, the one that is taking place after the the signal transition. In response, the **Tag** field shown in Figure in Figure 1-63 will display a signal transition on the selected signal line and the signal name will be deselected (turned off blue).

To set a **Tag** on a bus, first select the bus and then enter the desired bus state in the red **Bus State** window shown in Figure in Figure 1-65. Next, click on the blue **BUS** button.

To delete any signal condition within the **Tag**, click on the associated signal name and after it turns blue, click on the **Del** button shown in Figure in Figure 1-65.

Searching for tags

ACTIVE-CAD allows you to search for the **Tag** conditions both in the left and right directions. To select the **Tag** search mode, click on the **Search** button shown in Figure 1-25 till it displays the **Tag** mode (Figure 1-66). Next, click on the left and right pointing arrows to the left and right of the **Tag** button. Note that the blue vertical cursor moves to the next **Tag** condition in the selected direction.



Figure 1-66. Simulator's Search buttons.

Stopping long simulations

Since ACTIVE-CAD releases the mouse cursor when the simulation is in progress, you can click on the **STOP** button in the simulators main tool-box and terminate the simulation. When the simulation stops the design retains the last logical states that have been generated by the simulation process.

The **STOP** button terminates the simulation process independently of how it was started, either with the **Step** buttons or the **Start Long Simulation** option (in the **Options** menu).

NOTE: The **Stop** button can be activated at any time unless it is disabled in the **Start Long Simulation** window, activated by the **Simulation Stop** option in the **Options** menu.

Manipulating switches

ACTIVE-CAD allows you to manipulate switches and move jumpers while the simulation is in progress. Since the new switch position takes instant effect, ACTIVE-CAD operates like a real hardware breadboard.

To operate a switch, draw a schematic with a switch and select the **Switch Settings** option from the **Patching** menu. ACTIVE-CAD displays the **Switch Settings** window which shows the design hierarchy in the **Scan Hierarchy** window and lists all switches and jumpers at the selected hierarchical level in the **Chip Selection** window.

To manipulate a switch located at a selected hierarchical level, select that hierarchical level from the **Scan Hierarchy** window and then double-click on the desired switch in the **Chip Selection** window. ACTIVE-CAD instantly displays an additional window with the current switch position. Each click of the mouse cursor over the switch outline will toggle, rotate or move its mechanical wiper position.

The new switch position has an instant effect on the design behavior. After setting the switch to the desired position, close the setup window by clicking on the **Close** button.

Simulating line (layout) delays

ACTIVE-CAD can simulate line or layout delays. Normally, these delays are calculated by the layout tools and fed into ACTIVE-CAD via a netlist.

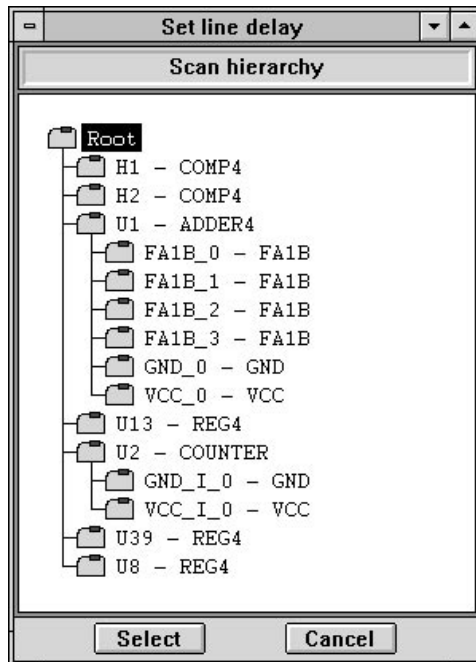


Figure 1-67. Modifying Line Delays.

You can view the line delays in ACTIVE-CAD only when the netlist includes such routing delays. The line delays are assigned to device I/O pins and can be viewed and edited in real time by selecting the **Change Line Delays** option from the **Patching** menu. ACTIVE-CAD responds with Figure 1-67 which lists the design hierarchy. Double-clicking on any chip, cell or macro in the hierarchical diagram of Figure 1-67 displays all layout propagation delays associated with the selected item (Figure 1-68). Note that the TpL and TpH for the same pin may have different layout delay values.

Name	Min	Ave	Max	Set	Description
U4.A - TpL	1.3ns	3.2ns	4.5ns	4.5ns	Entire
U4.A - TpH	1.4ns	3.4ns	4.7ns	4.7ns	Entire
U4.Y - TpL	0.0	0.0	0.0	0.0	Entire
U4.Y - TpH	0.0	0.0	0.0	0.0	Entire
U5.A - TpL	1.4ns	3.5ns	4.8ns	4.8ns	Entire
U5.A - TpH	1.5ns	3.7ns	5.2ns	5.2ns	Entire
U5.Y - TpL	0.0	0.0	0.0	0.0	Entire
U5.Y - TpH	0.0	0.0	0.0	0.0	Entire
U6.A - TpL	1.5ns	3.6ns	5ns	5ns	Entire
U6.A - TpH	1.6ns	3.8ns	5.3ns	5.3ns	Entire
U6.Y - TpL	0.0	0.0	0.0	0.0	Entire

OK Cancel % of Max

Figure 1-68. Layout Propagation Delays Table.

Clicking on **Min**, **Max** or **Avg**, transfers the values displayed in these columns to the **Set** column which represents the simulated values. You can directly edit the delay values in the **Set** column.

By entering a value into the **% Max** you can rescale the propagation delay in reference to the maximum parameters, as listed in the **Max** column.

NOTE: You must click on the **OK** button shown in Figure in Figure 1-68 to enforce any line delay modifications.

Emulating layout changes

Before you do any layout changes, you should verify what effect they will have on other cells and devices. If you have a general idea of how the layout changes will effect layout delays, then enter the expected delay values directly into the **Set** column.

If the design is failing because of timing problems, you can emulate the proposed layout modifications to see if they would cure the problems. If you have a good grasp of the physical limitations within your design, you can emulate various layouts by entering them directly into the **Set** column. Since editing the timing table is much quicker than modifying the actual chip or board layout, you save a lot of time by verifying the proposed layout changes with the real-time emulation process.

Presetting a design to the desired state

ACTIVE-CAD allows you to set the design to any logical state, including forced signal transitions. You must, however, remember that these set-

tings are forced directly on the output signal lines and they will be overridden with the devices actual output value after the first (simulation) transition within the device.

Manual design preset

To preset signal lines to the desired signal logical states, click on the **Selective Preset** option within the **Utilities** menu. ACTIVE-CAD responds with the window shown in Figure 1-69, which initially does not display any signal names. To select signals for the preset operation, click on the **Add** button, and select the desired pins and signal names from the new ACTIVE-CAD window.

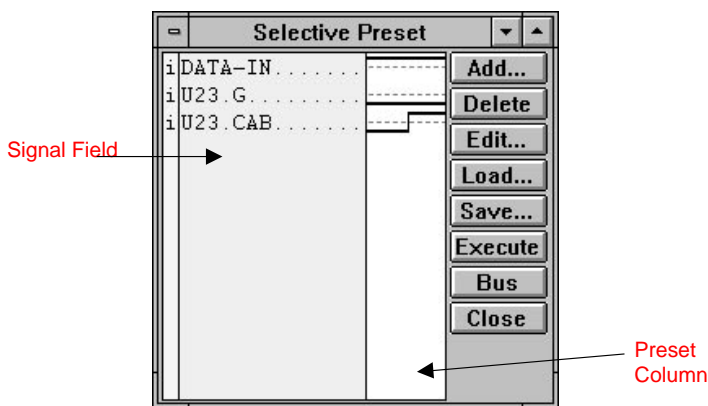


Figure 1-69. Preset Conditions Table.

After the device pins and signal names have been selected, click on the **Edit** button. A table with signal logical states (Figure 1-70) is displayed. Click now on the selected signal line shown in Figure in Figure 1-69 and then on the desired logical state shown in Figure in Figure 1-70. Note that the selected signal line has been assigned the new logical state in the preset column of Figure 1-69. You may also drag the selected logical state from the table in Figure 1-70 and drop it over the desired signal name in Figure 1-69.

To assign the same signal level to multiple signals, hold down the **[Ctrl]** key while selecting additional signal lines. Next, click on the desired logical signal state in the table of Figure 1-70. All selected signal lines will be assigned the same selected logical state.

To force a signal transition on a pin, e.g. from High-to-Low, first select the pre-transition preset state (e.g. High) and then select the post-transi-

tion state (e.g. Low). In response, ACTIVE-CAD will display in the preset column of Figure 1-69 the selected signal transition, e.g. High-to-Low.

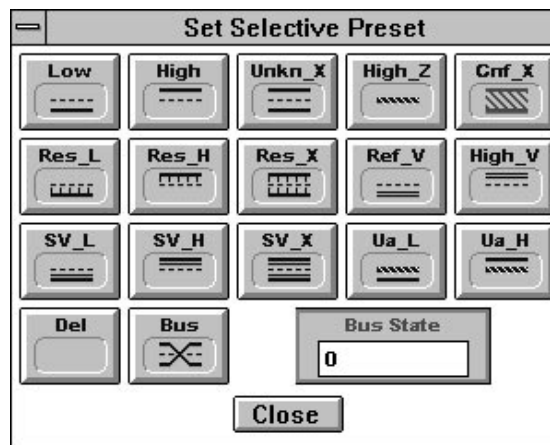


Figure 1-70. Selecting states for selective preset.

Automatic preset from a file

You can preset the entire design to selected signal conditions with a click of the mouse (ACTIVE-CAD only). That preset can be performed at any time during the simulation. To use the file preset option, you need to prepare in advance some preset files, as described in reference to Figure 1-69, and save these preset files by clicking on the **Save** button shown in Figure in Figure 1-69. Next:

- ☐ load the selected preset file by clicking on the **Load** button
- ☐ click on the **Execute** button to activate the presets
- ☐ click on the short **Step** or **Long** step button to simulate the preset conditions

Passive components in digital designs

Resistors always lower the signal strength. For example, test point A in Figure 1-71 has a strong signal. However, at test point B (other end of the resistor), the signal is weak. Thus, several totem-pole outputs can be connected together via resistors into a single node. Also, a totem-pole output can be connected through a resistor to the Vcc supply voltage without creating a conflict.

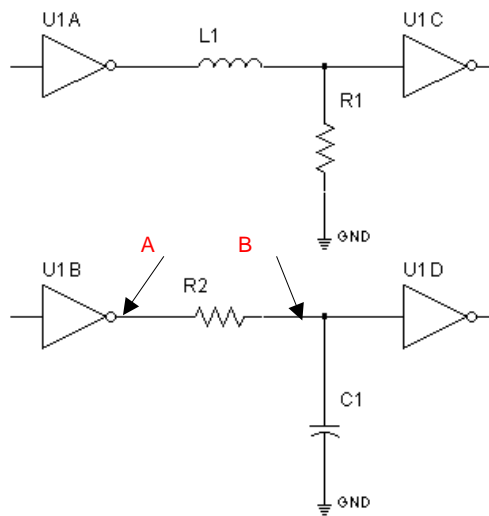


Figure 1-71. Passive Components Simulation.

The passive components are simulated as follows:

- ☐ resistor converts a strong signal into a weak one
- ☐ capacitor is at first a short circuit which opens after time Tc1
- ☐ inductor is at first an open circuit which closes after time Tc2

The time constants Tc1 and Tc2 can be set from the **Edit Timing Specification** window which is available from the **Patching** menu. Setting these time constants is identical to changing the propagation delay of typical digital devices.

NOTE: You cannot connect in series any passive components. For example, a serial connection of resistor and capacitor needs to be replaced with a capacitor alone. The capacitors time constant should be set within the **Edit Timing Specification** window.

Using multiple clocks in the design

Signal waveforms have a limited duration, as defined by their formula. Clocks on the other hand are repetitious signals that have no time limitation, except for the simulator test vector limitations. You can use the binary counter outputs as clock signals. However, if you need more complex clocks, define them using the following procedure:

Creating Clocks

- ☐ Select the **Stimulator** menu. Then select the **Add Stimulators** option.
- ☐ When the **Stimulator Selection** menu (keyboard) appears, select the **Formula** button
- ☐ In response, the **Set Stimulator** window appears
- ☐ Select the **Clocks** option from the **Mode** field
- ☐ Double-click on one of the selected clocks (**C1-C4**) in the **Select Clock** field
- ☐ Enter the clock formula into the **Formula** field
- ☐ Click on the **Assign Formula** button
- ☐ Select another clock from the **Select Clock** field and repeat the above two steps

Note that as you enter the clock formula, it is displayed in the **Defined Assignments** field in Figure 1-20. When you finish editing clock formulas, click on the **Close** button.

Applying Clocks

To apply multiple clocks to your design, follow this procedure:

1. Click on the selected signal in the **Signal** field of the Waveform Viewer window to mark the signal blue.
2. Select the **Stimulator** menu. Then select the **Add Stimulators** option.
3. When the **Stimulator Selection** window appears, click on the clock button (**C1-C4**), whose formula you want to apply. These buttons are located to the right of the keyboard display, and they are in black when defined. When gray, the clock buttons have not been assigned a formula.

You can assign the same clock formula to any number of signals. Up to four different signals, defined by formulas, can be assigned to the design. In addition, you can use any number of clock signals based on the binary counter outputs(B0-BF and N0-NF).

Simulating a Netlist

ACTIVE-CAD can work with schematic editors in real-time interactive mode (DDE) mode and off-line (netlist) mode. To select the operational

mode, select the **Project Manager** from the **File** menu. When the **Project Manager** window appears (Figure 1-72), select the **Netlist** option. In response ACTIVE-CAD displays the **Project Netlist Configuration** window shown in Figure 1-73.

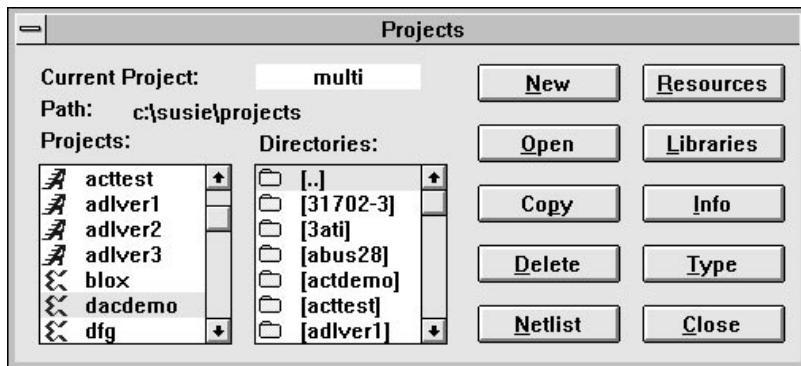


Figure 1-72. Project Manager window.

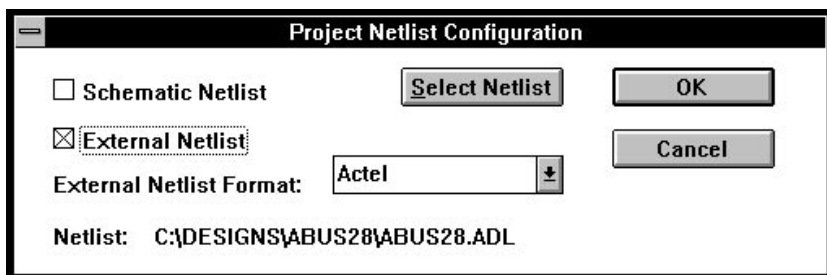


Figure 1-73. Netlist Configuration window.


Incremental netlist mode

If you click on the **Schematic Netlist** option shown in Figure in Figure 1-73, ACTIVE-CAD will communicate with the associated schematic editor via Windows DDE protocols. Only the initial design state is loaded via the schematic netlist. All additional schematic changes will be fed through the DDE functions. This allows an incremental netlist updating and enables the design to be modified while the simulation is in progress. To complete the netlist selection, click on the **OK** button.

Off-line netlist mode

If you click on the **External Netlist** option shown in Figure 1-73, ACTIVE-CAD highlights the **Select Netlist** button and the **External Netlist Format** field. To select the desired netlist format, click on the **External Netlist Format** field, select the desired netlist format and then click on the **Select Netlist** button. The simulator displays netlist selection window. Select the desired external netlist and to complete the netlist selection, click on the **OK** button in the **Project Netlist Configuration** window.

Flat netlist simulation

Flat netlists have all their components and connectivity explicitly listed. If you click on the **Design Contents**  button, ACTIVE-CAD will explicitly list all the components in the **Chip Selection** field (Figure 1-74). If that field lists any hierarchical macros, designated by Hxx, then the loaded netlist is a hierarchical one. The flat netlist displays only the Root symbol in the **Scan Hierarchy** level. To better understand the difference between flat and hierarchical netlist displays, compare the **Scan Hierarchy** fields shown in Figures 1-74 and 1-75. If a hierarchical design is flattened before it is loaded into ACTIVE-CAD, it will be listed as a flat netlist, similar to the one shown in Figure 1-74 (there is only **Root** in the **Scan Hierarchy** field).

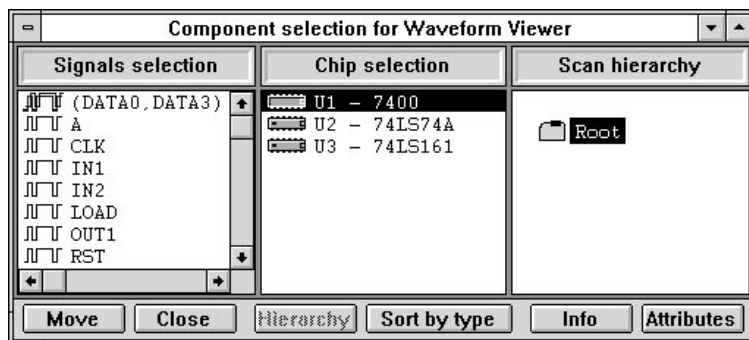


Figure 1-74. A Listing of Flat Netlist Components.

Hierarchical netlist simulation

If the **Chip Selection** field shown in Figure in Figure 1-75 lists hierarchical macros H1, H2, H3, etc., mixed with some concrete devices that are used at the design root level, then the design netlist is in the hierarchical format. The **Scan Hierarchy** window displays a hierarchical design struc-

ture and the **Signals Selection** field lists all signals that exist at the highlighted hierarchy level.

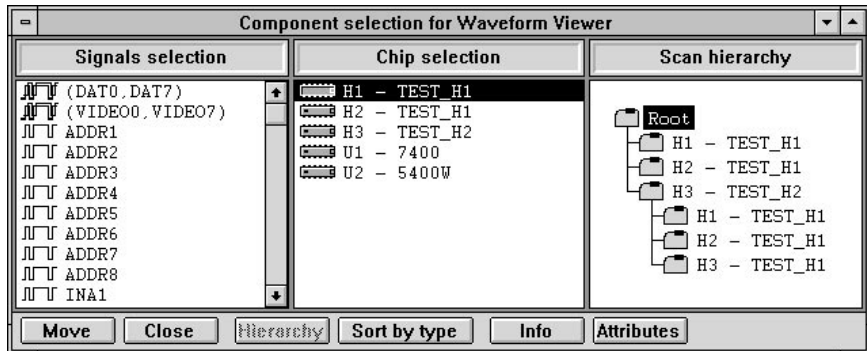


Figure 1-75. Listing of Hierarchical Netlist Components

If you review the ACTIVE-CAD internal netlist under a text editor, you will notice that it only contains the top level components and macros. All hierarchical drawings are listed as library components and their wiring and components are included within the hierarchical library models. For this reason, a hierarchical netlist may at first look as though it is missing a lot of components.

How To Save Selected Signal Names

Unlike batch-mode simulators, ACTIVE-CAD gives you instant access to any design section. You can create sets of signal files for testing selected design sections, and then test these design sections by loading the selected signal files to the screen.

To create a file made of selected signal names:

- ☐ Load the design netlist
- ☐ Select the **Add Signals** option within the **Signals** menu
- ☐ Select the desired signals from the new window
- ☐ Save these signals as a file (using **Save Test Vectors** or **Save ASCII Test Vectors** from the **File** menu).

NOTE: It is important to remember that you can save the signal names alone, without any associated waveforms.

Moving test vectors from one design to another

You can move any test vector files from one design to another even if the signal names do not match at all. This is very useful if you test certain design functions like counting, decoding, shifting, etc. To transfer test vector files from one design to another, follow this simple procedure:

- ☐ Save the desired test vector file as an ASCII file
(use the **Save ASCII Test Vectors** option in the **File** menu)
- ☐ Using a DOS or other editor, manually edit the saved ASCII file to contain the new signal names
- ☐ Save this file, preferably under a new name
- ☐ Load the new design netlist
- ☐ Use the **Load ASCII Test Vectors** option in the **File** menu to load the edited file

Switching between schematic and external netlist

ACTIVE-CAD allows you to dynamically disconnect any links to an on-line schematic and simulate an external netlist. To switch from an on-line schematic editor to an external netlist, select the **Netlist** option from the **Project Manager** window, which is selectable from the **File** menu. When the **Project Netlist** window shown in Figure 1-73 appears, click on the **External Netlist** option and select the appropriate netlist format and file (see **Loading Netlists**).

To switch from an external netlist to the on-line schematic editor, click on the **Schematic Netlist** option (Figure 1-73) and the **OK** button. ACTIVE-CAD will load the currently active schematic design into the simulator and all your subsequent design changes will be fed incrementally.

Tracking Errors Through a Design Netlist

ACTIVE-CAD allows you to track design errors directly from the netlist signal state display. When an error occurs, you can put a number of test points around the suspect devices and along the signal paths and trace the origin of the design problem. A quicker way to track a design problem is by tracing the signal lines and their states through the netlist. Example 12 shows a typical signal tracing process through the design netlist.

Example 12:

Load the *TEST_A2* binary test vector file (Figure 1-76), which is associated with the *TEST_A* schematic. Simulate one **Long** step; note that the *OUT1* signal line shows the X_Unknown logical state.

To trace the origin of this X_Unknown state, click on the *OUT1* signal name shown in Figure in Figure 1-76 and then select **Connections** from the **Signal** menu. ACTIVE-CAD responds with Figure 1-77, which shows that the *OUT1* is driven by the U2(74LS74A).Q1 output pin.

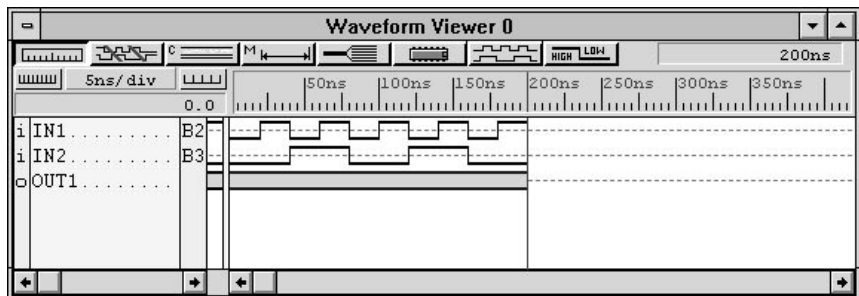


Figure 1-76. *TEST_A2* Test Vectors after the long step.

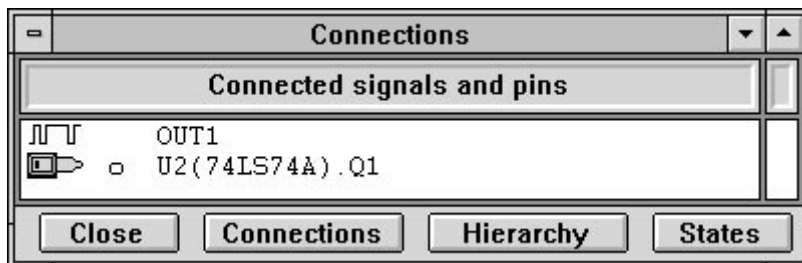


Figure 1-77. *OUT1* Signal Connections Listing.

To analyze in detail the U2(74LS74A).Q1 output pin logical state status, click on the **States** button shown in Figure 1-77, which forces ACTIVE-CAD to display the signal logical states. In response, ACTIVE-CAD displays Figure 1-78 which lists:

- ☐ **Node** - the resulting signal in the signal node
- ☐ **Conv** - how the model will interpret the input signal (inputs only)
- ☐ **Model** - what the model will output (outputs only)

- ☐ **Stim** - shows manually applied pin/node override signal



Connections					
Connected signals and pins		Node	Conv	Model	Stim
	OUT1	Unkn_X	Unkn_X	----	----
	U2(74LS74A).Q1	Unkn_X	----	Unkn_X	----
Close		Connections		Hierarchy	
				States	

Figure 1-78. OUT1 Logical States Table.

To understand how the X_Unknown state has been generated at the U2(74LS74A).Q1 pin, let's analyze its inputs (you need to know how the 74LS74A operates). First, double-click on the U2.Q1 output. When ACTIVE-CAD displays the list of all device input pins, double-click on the U2(74LS74A).CLK1 signal line. In response, ACTIVE-CAD displays the window in Figure 1-79 which confirms that the U2.CLK1 signal line has no stimulus signal and remains in the High_Z state. Clicking on the signal lines U2.PRE1 and U2.CLR1, produces a similar result.



Connections					
Connected signals and pins		Node	Conv	Model	Stim
	CLK	High_Z	----	High_Z	----
	i U2(74LS74A).CLK1	High_Z	High	----	----
Close		Connections		Hierarchy	
				States	

Figure 1-79. CLK Node Logical States Table.

Conclusion: The U2 device is in the unknown state because all input signals are in the High_Z state and U2 has not been reset after power-on.

Should there be a problem with the U2(74LS74A).D1 input signal, you could double-click on this pin. This would list the logical states of all signals in the U2.D1 node (Figure 1-80).




Connections					
Connected signals and pins		Node	Conv	Model	Stim
	1 A	High	----	----	----
	U1(7400).Y1	High	----	High	----
	i U2(74LS74A).D1	High	High	----	----
Close		Connections		Hierarchy	
				States	

Figure 1-80. Logical States Table for U2.D1 Pin.

Typically, you need only look at the output pins in the node because they control the signals in each node. Thus, double-clicking on the U1(7400).Y1 output would produce a listing of the gate inputs. The process of signal tracing may continue, as described in reference to the U2.D1 input.

The **Connections** option has been expressly developed for the ALDEC technical support staff so they can quickly and effectively trace any design problem that they may receive from around the world. Once you become familiar with this process, you you will be able to trace your design problems with an insight that no other simulation tool can provide.

How To Simulate FPGA Designs At The System Level

To accommodate various budgetary restrictions, ACTIVE-CAD products come in two basic configurations:

- ☐ chip-level design tools, and
- ☐ system-level design tools

Chip-level design tools

These low-cost tools allow you to simulate a single device at a time. Each project is a separate FPGA or ASCII design which you can enter and troubleshoot at pre-layout (functional) and post-layout (timing) design cycles.

All ACTIVE-CAD products allow you to perform functional system-level analysis, based on the pre-layout design files. The process of functional system level simulation is comprised of the following steps:

- ☐ save each FPGA and ASIC as an independent macro
- ☐ create a top-level design with these macros included
- ☐ load this top-level design netlist into the logic simulator

The chip-level ACTIVE-CAD allows only one post-layout netlist to be loaded but does not care if you load several pre-layout netlists as part of the same design file.

System-level design tools

The system-level design tools allow you to simulate multiple FPGA and ASIC chips at the board level. Since these ACTIVE-CAD products also simulate the post-layout designs, they produce very reliable design data.

To use the board-level simulation capabilities most effectively, create a schematic design with ASIC and FPGA packaging information. Next, connect or wire these packages together. Following this, assign to these packages concrete FPGA or ASIC netlists. If you are using the ACTIVE-CAD (DDE-related) schematic, select the **Assign Netlist** option from the **Hierarchy** menu. And when the netlist window appears, select the desired netlist and assign it by clicking on the selected package on the schematic.

Chapter 2

ACTIVE-CAD Logic Simulator

Using The ACTIVE-CAD simulator

Simulator Window

The ACTIVE-CAD simulator window contains the title bar, menu bar and control bar, all located at the top of the screen. Since ACTIVE-CAD can handle multiple windows, items such as signal waveforms, components, timing parameters¹, etc. are all displayed in separate windows. The simulator main toolbox allows you to switch between the functional, glitch and timing simulation modes. It also has provisions for running simulation steps, searching for tags, breakpoints and other options.

All key elements of the simulator window are shown in Figure 2-1.

1 **NOTE:** The timing simulation mode is not included in SUSIE-CAD/HIER.

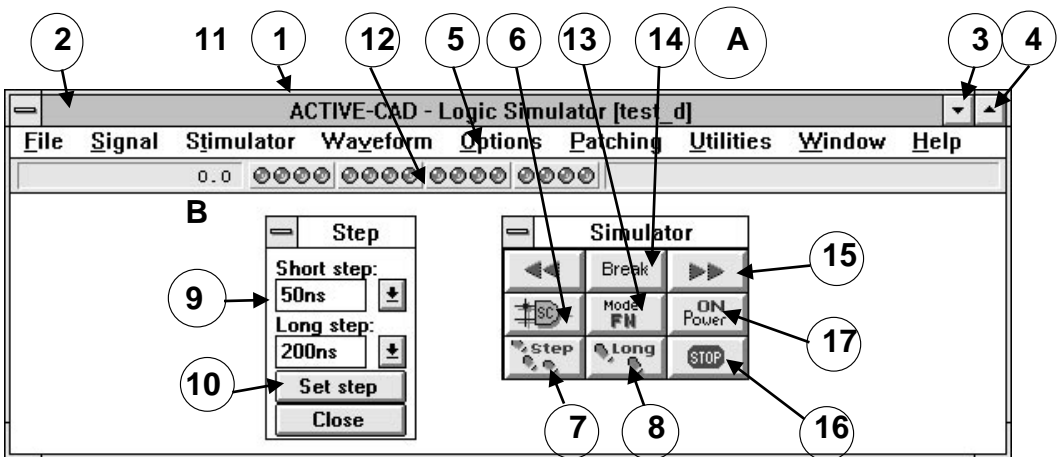


Figure 2-1. Simulator Main Window.

#A. **Main toolbox** - provides quick access to the most important simulator operations. The toolbox can be moved to any screen location and can be displayed using the **View/Main Toolbox** option in the **Utilities** menu.

#B. **Step window** - used to set up the simulation steps. It can be moved around the screen. To display this window select the **Simulation Step** option in the **Utilities** menu.

#1. **Title Bar** - used to display the projects name. It is also used to move the simulator window around the CRT screen. To move the simulator window, click on the title bar and drag the window to the desired screen location. When the simulator window has been placed at the desired screen location, release the mouse button. If you double click on the title bar, it will toggle between regular and full screen window size.

#2. **System menu** button - invokes a typical Windows menu, with options like Close, Maximize, Minimize, Move, etc.

#3. **Minimize** button - allows you to reduce the entire window (minimize) to a small icon that will be placed at the bottom of the screen. To restore the window to its full size, you need to double-click on the minimized window icon.

#4. **Maximize** button - used to display the window over the entire screen. To restore the window to its previous size, toggle the upper right button of the maximized window.

#5. **Menu bar** - includes eight menus which can be accessed by clicking on the menu name or by pressing the **Alt** key and the underlined character in the menu name.

#6. **Schematic editor** button - switches the screen to the ACTIVE-CAD schematic editor program. If the schematic editor has already been started, ACTIVE-CAD will display its window. Otherwise the schematic editor will be started.

#7, #8. **Short** and **Long Step** simulation buttons - used to perform short and long simulation steps with a single click of the mouse button.

#9, #10. **Short** and **Long Step Setup** - used to set up the **Short** and **Long Step** duration. You can either click on the field and enter a new value, or click on the arrow to access a list of predefined simulation steps. This setup window pops up when you select the **Simulation Step** option from the **Utilities** window.

#11. **Simulation time** - displays the length of the current simulation cycle in nanoseconds, counting from the beginning of simulation. The maximum simulation time depends on the simulation resolution. For 10ps resolution the maximum time is 42ms. For other resolutions the simulator may simulate minutes and hours of hardware operational time.

#12. **Binary counter** - the status of the 16 bit binary counter is displayed in the form of red and yellow LED lights. Red means that the bit is at a logical high or 1, and yellow means a logical low or 0.

#13. **Simulation mode setup** - allows you to select between **TM** - timing mode with a 10 picoseconds resolution, **GL** - glitch mode with **Short** step unit propagation delays, **UN** - unit delay mode with propagation unit equal to simulation precision and **FN** - functional mode with zero propagation delays. The **TM** option is not available in SUSIE-CAD/HIER.

#14. **Search** button - allows you to select an item or condition to search for in a timing diagram. You can choose from such items as Breakpoints, Errors, Events, Milestones and Tags.

#15. **Search Forward** and **Backward** buttons - allow you to search timing areas for items you have selected with the **Search** button (#14).

#16. **Stop Simulation** button - stops simulation at the current simulation cycle. It is enabled and disabled within the window resulting from clicking on the **Simulation Stop** option within the **Options** menu.

#17. **Power On** button - initializes the entire design. The Power On operation is performed automatically at the beginning of simulation and after each interactive connectivity change.

Waveforms Window

A typical waveform window is shown in Figure 2-2. Using the **Utilities** menu, you can open multiple waveform windows, also called Waveform Viewers. The signals displayed in all windows will be the same; However, you can view different parts of the waveform diagram and use different timing scales. If you minimize these windows, the icons will be placed at the bottom of the main window.

NOTE: The setups and controls located within each Waveform Viewer window effect only that window.

The Waveform Viewer window has the following main areas:

Signal entry - comprised of fields #13 and #14; field #13 displays the I/O function of the listed signals and IC pins, and field #14 displays the signals and IC pin names (to be observed or stimulated with a test vector)

Test vector entry - includes field #17 for entering ready-made test vectors and field #10 for direct test vector editing. The current logical state of all signals under the blue vertical cursor is displayed in field #18.

Waveform window setups - includes the Scale display (#16) of the waveform window, waveform cursor position (#11) and a tool bar for managing the signals and test vectors (#1 through #8).

ACTIVE-CAD has two vertical cursors displayed in the signal waveform area. The red cursor is the simulation cursor. It is visible at all times and indicates the last simulation cycle. The blue cursor is the editing cursor and is visible only if you click at any screen location for editing or viewing the local simulation data. The logical states under the blue cursor are explicitly listed in column #18.

Zoom In and **Zoom Out** - these buttons (#15) are used to quickly expand and contract the waveform scale. Each mouse click on one of these buttons changes the scale to the next predefined value.

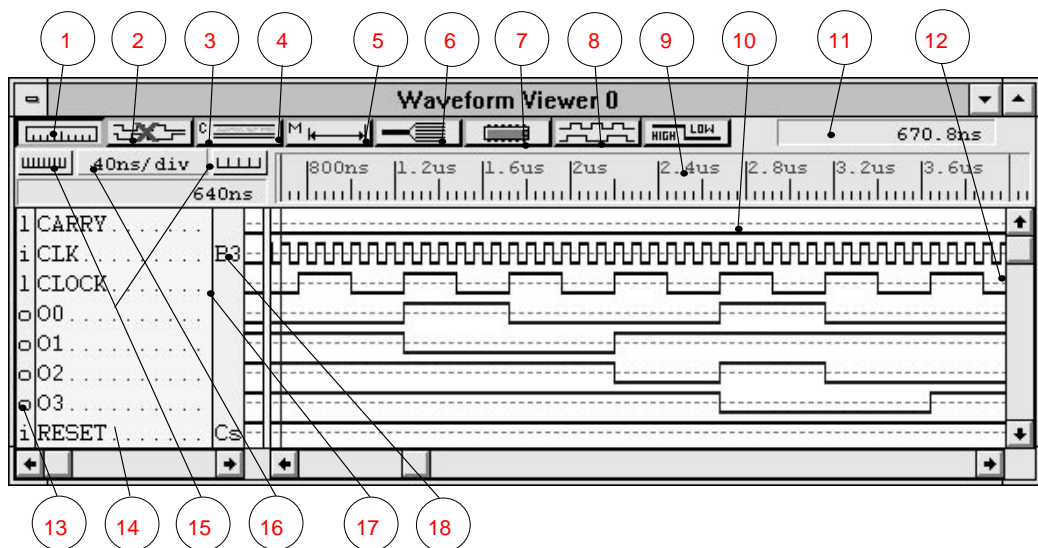


Figure 2-2. The Timing Window.

The waveform window tool bar has the following function buttons:

- #1. **Ruler On/Off** - enables or disables the waveform window ruler. If you don't need to see the waveforms referenced to the time scale, disable the ruler to allow more room for signals.
- #2. **Waveform Delete** - deletes all waveforms without resetting simulation with Power On
- #3. **Display Comments On/Off** - allows you to toggle the comment display on and off. The comments are used to document important situations on the waveform diagram and can be both displayed at the specified screen locations and printed with the waveform diagram.
- #4. **Measurements On/Off** - enables you to display precise timing measurements between signal transitions, regardless of the scale.
- #5. **Bus On/Off** - clicking on this button is meaningful only if you have defined some buses, either in Signal entry (#14) or on the schematic. Buses are comprised of several signals or pins and can be displayed in hex, binary, decimal and octal mode. When the bus display is set to the OFF mode, each bus signal line is individually displayed.

#6. **Select Probes** - invokes the **Component Selection** for Waveform Viewer window which allows you to select signals and IC pins for display in the Waveform Viewer window.

#7. **Stimulus** - invokes the **Stimulator Selection** window that is used to define and assign stimulators or test vectors to the selected signals. This window includes such signal waveform generators as binary counter, keyboard keys, asynchronous clocks and waveform formulas.

#8. **Logical States** - invokes the **Stimulator State Selection** window that allows you to select and assign any logical state to a signal name or device pin. The new logical states can be assigned to input pins and signals at any time during simulation.

#9. **Time Scale Display and Expansion (Ruler)** - It has a dual purpose:

- ☐ It displays the screen resolution. The scale can be used as a general reference for the simulation results (waveforms).
- ☐ If you click on the scale and drag the mouse cursor, a blue stripe follows the mouse movement. If you release the mouse button, the area under the blue stripe expands to the full screen display. It is used for observing alignment of signal transitions and time measurements.

#10. **Signal Waveform Display** - displays the test vectors that stimulate the design and real time responses from the design.

#11 **Blue cursor location** - displays the current location of the blue cursor which is used for direct waveform editing, time measurements, comments insertion, etc. It is automatically hidden under the red cursor, immediately after the first simulation event.

#12. **Red cursor location** - shows the last simulation cycle.

#13. **I/O Attribute** field - shows the I/O type of the signal (i=input, o=output, b=bi-directional).

#14. **Signal** field - displays the names of the signals and device pins which have been selected for display.

#15. **Scale Adjustment** field - clicking on the left-hand field increments the display scale of the signal waveforms, e.g. from 1 ns to 2 ns per division. Clicking on the right-hand field reduces the display scale of the signal waveforms, e.g. from 5 ns to 2 ns per division.

#16. **Scale Display** - shows the scale of the signal waveform display.

#17. **Stimulator** field - allows to enter the names of the pre-defined stimulators. The assigned stimulators will control the associated signal lines.

#18. **Current Logical State** - displays the logical states of signal waveforms at the current blue cursor location.

ACTIVE-CAD Main Menus

The ACTIVE-CAD operation is controlled from a set of menus, which load design, select test points, feed design stimulus and set the design analysis environment. A brief description of each menu follows below. For higher productivity, make yourself familiar with the features available in each menu.

Since most of the Windows operations are self-explanatory, key attention will be paid to ACTIVE-CAD specific features, options and operations. Some ACTIVE-CAD features may be invoked from several applications. To minimize cross-referencing, these features will be described in detail in several chapters.

FILE Menu

The **File** menu allows you to load, save and print selected design files:

- | | |
|---|---|
| <input type="checkbox"/> Load Test Vectors | loads binary files with design stimulus |
| <input type="checkbox"/> Load ASCII Test Vectors | loads ASCII files with design stimulus |
| <input type="checkbox"/> Save Test Vectors | saves signal waveforms in binary files |
| <input type="checkbox"/> Save ASCII Test Vectors | saves signal waveforms in ASCII files |
| <input type="checkbox"/> Load Memory Chip | loads memory chip with hex data |
| <input type="checkbox"/> Save Memory Chip | saves the contents of a memory chip |
| <input type="checkbox"/> Load Memory Block | loads memory block with hex data |
| <input type="checkbox"/> Save Memory Block | saves the contents of a memory block |
| <input type="checkbox"/> Load Fuse Map | loads a fuse map into a selected PLD device |
| <input type="checkbox"/> Load Simulation | reloads a previously simulated design |
| <input type="checkbox"/> Save Simulation | saves the simulation results for future use |
| <input type="checkbox"/> Page Setup | typical Page Setup operation |
| <input type="checkbox"/> Print Setup | typical Print Setup operation |
| <input type="checkbox"/> Print | typical Print operation |
| <input type="checkbox"/> Print Error Report | prints a design error report |

- | | |
|---|---|
| <input type="checkbox"/> Project Manager | selects the project and its resources |
| <input type="checkbox"/> Project Libraries | selects device libraries for simulation |
| <input type="checkbox"/> Load Netlist | loads a design (netlist) |
| <input type="checkbox"/> Test PLD | allows a quick, single PLD simulation |

SIGNAL Menu

The **Signal** menu allows you to select signals to which you will apply stimulus signals or which you will analyze in detail:

- | | |
|---|--|
| <input type="checkbox"/> Add Signals | has extensive sub-menus for selecting signals and pins |
| <input type="checkbox"/> Hierarchy | allows you to select signals/pins from any hierarchical level |
| <input type="checkbox"/> Connections | allows you to track signal paths and their logical values |
| <input type="checkbox"/> Move to | allows you to rearrange signals on the screen |
| <input type="checkbox"/> Bus | selects, creates and enables buses for display |
| <input type="checkbox"/> Delete | deletes signals and buses from display |
| <input type="checkbox"/> Select | selects and deselects signals for further processing |
| <input type="checkbox"/> Insert Empty Line | inserts an empty line at the current cursor location |
| <input type="checkbox"/> Search | searches for the selected signal/pin name in the display screen |
| <input type="checkbox"/> Find... in SC | locates selected signal/pin names on the DDE-connected schematic sheet |
| <input type="checkbox"/> Signal Set | allows you to define an ASCII set of test vectors |

STIMULATOR Menu

The **Stimulator** menu allows you to select, create and apply design stimulus signals, which are also called test vectors:

- | | |
|--|---|
| <input type="checkbox"/> Add Stimulators Mode | allows you to create and apply stimulus signals from a special window |
| <input type="checkbox"/> Chip Controlled | gives control of signal lines to device pins |
| <input type="checkbox"/> Override Mode | allows the stimulus signal to override a device output pin |

- | | |
|--|--|
| <input type="checkbox"/> Disconnect | disables the activity of a stimulus signal |
| <input type="checkbox"/> Connect | enables the stimulus signal activity |
| <input type="checkbox"/> Delete | deletes the selected stimulus signal |
| <input type="checkbox"/> Delete All | deletes all assigned stimulus signals |

Waveform Menu

The **Waveform** menu allows you to edit existing stimulus signals, add comments and perform waveform timing measurements:

- | | |
|--|--|
| <input type="checkbox"/> Edit | enables you to graphically edit signal waveforms |
| <input type="checkbox"/> Comments | allows you to enter comments with each signal waveform |
| <input type="checkbox"/> Markers | allows you to set reference points on signal waveforms |
| <input type="checkbox"/> Measurements | facilitates time measurements between signal transitions |
| <input type="checkbox"/> Formula | allows you to express signal waveforms as sets of nested data, e.g.. <code>((H20nsL30.5ns)25...</code> |

OPTIONS Menu

The **Options** menu provides direct access to the ACTIVE-CAD options that simplify and speed design analysis:

- | | |
|---|---|
| <input type="checkbox"/> Selective Simulation | instantly selects the desired design sections for simulation |
| <input type="checkbox"/> Power On Settings | sets the logical states of device pins during power-on |
| <input type="checkbox"/> Clock Settings | sets the clock speed of the binary counter-driven stimulus signals |
| <input type="checkbox"/> Simulation Precision | sets the ACTIVE-CAD simulation precision (from 10 ps to 1 ms) |
| <input type="checkbox"/> Simulation Stop | selects simulation time and enables the STOP button |
| <input type="checkbox"/> End of Step Estimation | enables the display of time till end of simulation |
| <input type="checkbox"/> Timing Automatic Backup | automatically saves simulation data, as set in the menu |
| <input type="checkbox"/> Milestones | sets milestones along the waveforms for automatic return to past cycles |

<input type="checkbox"/> Global Reset	triggers the global reset function in IC models
<input type="checkbox"/> Transport Delay	propagates short pulses without filtering
<input type="checkbox"/> Error Reporting	controls which errors are reported, displayed and saved
<input type="checkbox"/> Memory Range	sets the simulation areas of the memory models
<input type="checkbox"/> Set Tag Condition	sets signal conditions that ACTIVE-CAD will search for
<input type="checkbox"/> Delete Tag Condition	deletes the Tag condition
<input type="checkbox"/> Save Settings Now	saves the current Windows settings
<input type="checkbox"/> Save Settings On Exit	saves the Windows settings upon exiting from the program
<input type="checkbox"/> Default Settings	loads factory-set ACTIVE-CAD features and options

PATCHING Menu

The **Patching** menu allows dynamic modification of the design timing parameters, including replacement of parts, changing propagation delays of ICs and layout, etc.:

<input type="checkbox"/> Change Technology	allows you to instantly replace parts with faster or slower ones
<input type="checkbox"/> Switch Settings	allows you to rotate switches, move jumpers, etc.
<input type="checkbox"/> Change Line Delays	allows you to make manual changes in layout delays
<input type="checkbox"/> Edit Timing Specification	allows you to manually edit device timing parameters
<input type="checkbox"/> Change Generic values	permits you to change the operational temperature of models

UTILITIES Menu

This menu is a collection of ACTIVE-CAD supporting utilities like the VHDL Development System, the batch simulation macro editor, the fault simulator, the memory configurator. It also lets you control several display options. The following is description of the menu contents:

<input type="checkbox"/> View	lets you control display of ruler, status line, Tags, main toolbox, etc.
<input type="checkbox"/> Macro	allows you to edit and run macros that control batch mode simulation
<input type="checkbox"/> Waveform Viewer	displays additional signal waveform windows
<input type="checkbox"/> Selective Preset	allows you to set design preset conditions
<input type="checkbox"/> VHDL Debugger	allows you to run the simulation at the source code level
<input type="checkbox"/> Internal State Editor	lets you view model internal states
<input type="checkbox"/> Memory Configurator	lets you set the memory chip configuration
<input type="checkbox"/> Memory Editor	permits you to edit memory contents
<input type="checkbox"/> Error Viewer	displays error reports on screen or printer
<input type="checkbox"/> Hierarchical Viewer	displays the design hierarchy
<input type="checkbox"/> Fault Simulator	activates the toolbox for fault simulation control
<input type="checkbox"/> Simulation Step	allows you to set the short and long simulation step durations
<input type="checkbox"/> Breakpoints	detects the desired signal states and their combinations, and allows you to perform selective simulator operations depending upon these signal states

WINDOW Menu

This is a typical Windows menu, as described in the Windows manual.

HELP Menu

Operation of this option is similar to other Windows programs. Starting with REV. 2.0 it will provide additional information about the available options at each cursor location.

Loading a Design

Board-Level Netlist

Working with the on-line schematic editor

When you start the simulator, the schematic design information is automatically passed to the simulator in the form of a netlist file. The ACTIVE-CAD netlist is a binary file that contains all component and connectivity information. The netlist file has an *.ALB file name extension and is stored with other project files. Managing netlist files is simplified by the **Project Manager** which controls the netlist file location and keeps the netlist file up-to-date. If the netlist file is older than the schematic, a new netlist is automatically created and loaded into the simulator.

The netlist is loaded only when you start or open the simulator window. Thereafter all schematic changes are incrementally updated in the simulator so that you always simulate the design that is displayed in the schematic window.

Working with off-line schematic editor

To simulate designs edited in OrCAD, P-CAD, PADS, Tango, Protel and other editors, you need to work with the ACTIVE-CAD simulator alone. To start the simulator alone, you must:

- ☐ enter into the **Project Manager** the name of the project that will be created when you load the external netlist
- ☐ select in the **Project Library** the model libraries that will be needed for the netlist design.

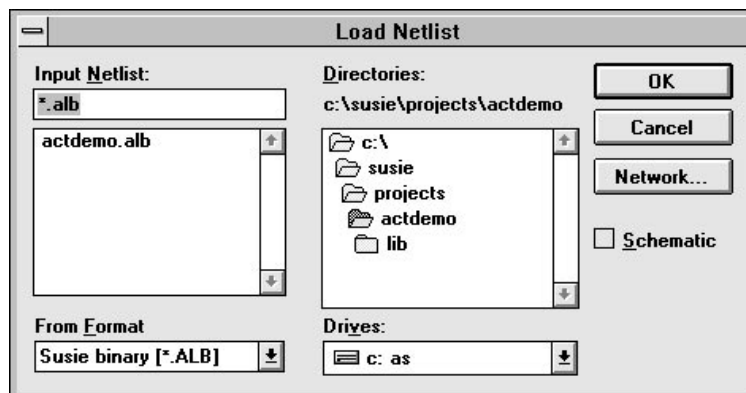


Figure 2-3. Loading netlist in the specified format.

If you are using an external schematic editor, you have to use its software subroutine and generate a netlist, which is an ASCII representation of the schematic design. Some schematic capture packages have facilities for generating multiple netlist formats such as EDIF, FutureNetTM, etc. In addition, some schematic capture packages can generate both flat and hierarchical netlist formats.

To load an external netlist to the simulator, use the **Load Netlist** option in the **File** menu.

When the window shown in Figure 2-4 appears, select the appropriate netlist format in the **From Format** field and then select the file name of the desired netlist from the list of files with that format. When you press the **OK** button, ACTIVE-CAD will convert and load the selected netlist to the simulator.

Hierarchical Netlist

ACTIVE-CAD simulates both flat and hierarchical netlist formats. When simulating hierarchical netlists, the signal, IC and IC pin names are the same as listed at the selected hierarchy level and they do not have any prefixes related to the upper design levels. They form totally independent netlists, which may list identical part numbers, e.g., U1, A5, etc.

A hierarchical netlist lists only its top-level (root) connectivity. The hierarchical macros have their own separate netlist files. A hierarchical project netlist is thus comprised of all these netlist files.

Combining Design Netlists

ACTIVE-CAD can simulate a design that is comprised of different netlist formats. For example, a board-level design can be in the FutureNet netlist format, while an FPGA design may be in the Xilinx XNF netlist format. Use the ACTIVE-CAD schematic editor to import these netlists:

- ☐ Import the system-level netlist using the **Import Netlist** option in the ACTIVE-CAD **Hierarchy** menu, within the schematic capture window
- ☐ Select **Assign Netlist** from the **Hierarchy** menu and click on the empty symbol to receive the imported netlist. Follow the prompts for loading the netlist.

Loading a Netlist

The simulator can be automatically invoked from any program with a command line. The command line allows you to specify the project to be loaded, or the netlist file to be converted and loaded.

Command line syntax:

```
SIMUL <-n | -p> <ID> <NAME> <PATH>
```

You can load designs either in the **-n**(netlist) or **-p**(project) format. Select one of these options for loading a netlist.

-p parameter loads the specified project and its netlist. This option can be used when the netlist has been previously imported and an ACTIVE-CAD project is ready for simulation (no changes on the schematic).

Example: *SIMUL -p TEST1*

This Example loads the project *TEST1* netlist in the ACTIVE-CAD format.

-n option is only used when you load a netlist in one of the supported external formats, like OrCAD, PADS, P-CAD, etc.

The **ID** parameter that follows is the format number from the list below:

<u>Netlist</u>	<u>ID</u>
SUSIE 5.0	0
Application Bravo	1
Cadnetix	2
CapFast	3
Case Technology	4
Design Computation	5
FutureNet	6
Ovation	7
OrCAD	8
P-CAD	9
P-CAD Hierarchy	10
PDIF	11
Racal-Redac	12
Tango	13
Visionix	14
Viewlogic	15
Wintek	16
SUSIE 6.0	17
XNF	18
EDIF 2.0	19
SUSIE binary	20
Lattice SIM	21

Massteck ASCII	22
PADS	23
INTUSOFT	24
SPICE	25

The netlist name is the next parameter to be entered (<NAME>).

You must also provide the directory in which this netlist resides (<PATH>).

To enter a command option, click on the simulator icon in the **Program Manager** group and select the **Properties** option in the **File** menu of the *Windows Program Manager*. In the **Command** line, add the desired parameters to the existing line and click on the **OK** button.

Example:

SIMUL -n 6 test6.net C:\NET

This will automatically convert and load the *test6.net* netlist in the FutureNet format (ID #6), located in the C:\NET directory.

Missing Models

Before loading a netlist, you have to select **Project Libraries** from the **File** menu and then select the required libraries.

While loading a netlist, the simulator automatically loads the integrated circuit (IC) models referenced in that netlist. If there is no model for a device, ACTIVE-CAD creates an empty model so that the simulation can proceed.

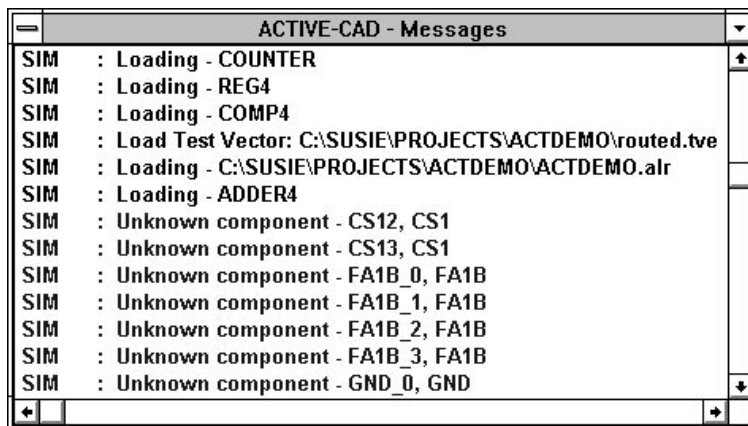


Figure 2-4. Missing Models Message.

The empty model is created based upon the schematic netlist and consists of device pins that are connected to other devices. For example, if the netlist has a device with 14 pins, but only pins 1, 2 and 7 are connected, then the simulator will create a model that has only pins 1, 2 and 7.

The outputs from the missing models are floated (high impedance state) during simulation and they do not interfere with other signals in the nodes. You can feed into the missing model outputs some test vectors that emulate the operation of the part itself. The simulation will proceed as though these test vectors had been generated by the empty model itself. The ACTIVE-CAD simulator will generate a precise system level response to these signals.

After loading a netlist, the simulator reports all missing models in the window shown in Figure 2-4. The **Component Selection** window shows empty models as white chip icons, which have pin numbers but no names.

Loading a PLD Fuse Map

When fitted with the PLD libraries, ACTIVE-CAD simulates PALs and PLDs using the standard JEDEC fuse maps. Each PLD model represents a complete but unprogrammed structure that is dynamically re-configurable by the loaded JEDEC file.

Typically, PLD device behavior can be monitored only at the external pins. However, some of the more complex PLDs like MACH devices have predefined internal test points (signals or buses) that you can load and watch on the screen.

After loading a netlist, load the JEDEC file into each PLD you want to simulate. The process of loading JEDEC fuse maps is as follows:

- ☐ select the **Load Fuse Map** option in the **File** menu
- ☐ when the simulator displays a list of all available PLDs (Figure 2-5) at the current hierarchy level, select the desired PLD
- ☐ then load the appropriate JEDEC file from the window shown in Figure 2-6, which appears when you select the PLD device.

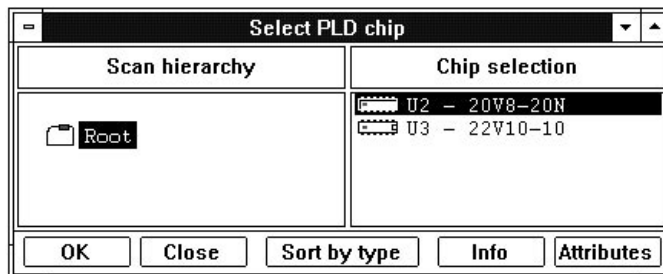


Figure 2-5. Selecting a PLD to load JEDEC file.

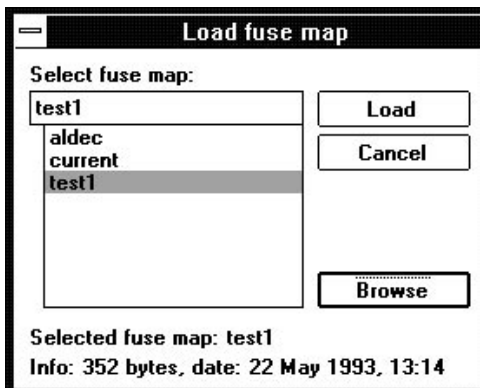


Figure 2-6. Load JEDEC File Window.

ACTIVE-CAD does not support editing of the fuse maps, but you can reload a different JEDEC file at any time.

To load a JEDEC file for the first time to the current project you may need to use the **Browse** option (Figure 2-6). This will allow you to find your JEDEC file on the appropriate disk, and add it to the project resources. You can also use the **Resources** option within the **Project Manager** window to prepare the list of project resources in advance.

Loading JEDEC fuse maps into hierarchical designs

To load a JEDEC fuse map into a PLD that resides within a schematic device (hierarchical schematic), select the **Load Fuse Map** option from the simulator **File** menu. ACTIVE-CAD will display the window in Figure 2-5, which shows the design hierarchy. Click on the desired schematic macro in the **Scan Hierarchy** window and when the PLD devices are listed to the right, in the **Chip selection** window, double-click on the desired PLD device. ACTIVE-CAD instantly displays Figure 2-6 which lists all available JEDEC fuse maps. Select the desired fuse map and click on the **Load** button. The PLD is ready to simulate. You can also activate the **Browse** option and load JEDEC files from other system directories.

Test PLD option

If you are designing a PLD, you can simulate it directly with ACTIVE-CAD without creating a schematic. To simulate multiple PLDs, you need to create a schematic with the appropriate connections between devices and then load all PLDs with their JEDECs.

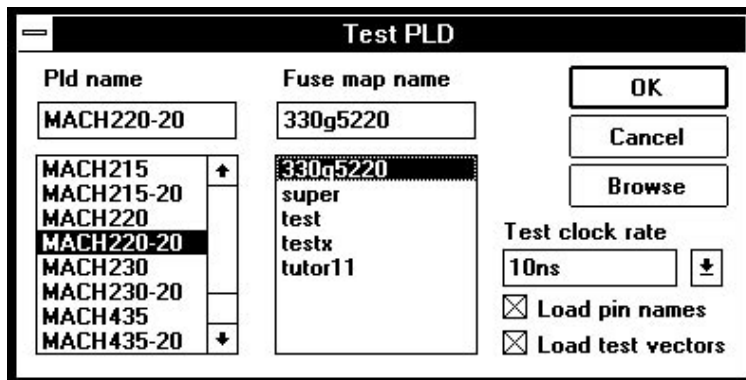


Figure 2-7. The Test PLD Window.

The **Test PLD** option in the **File** menu can also retrieve the pin names used in your design and load any test vectors that were saved in the JEDEC file. This allows you to compare the functional simulation from your PLD development system with the ACTIVE-CAD timing simulation. To perform this comparison, first load the test vectors from a JEDEC file into ACTIVE-CAD (you will see the original waveforms generated by the PLD development system). Next, start the simulation by pressing the **Short** step button, and visually compare the new waveforms that have been generated by ACTIVE-CAD with the ones loaded from the JEDEC file. This is especially useful when simulating a PLD in the timing

mode, because you will be able to instantly see how the propagation delays have changed the design behavior.

Selecting a PLD Device

To load a PLD design, select the **Test PLD** option in the **File** menu. When the **Test PLD** window (Figure 2-7) appears, select the PLD type you want to load. You can enter the full PLD name or any part of the name to search for. Use asterisks, e.g., *22v10*, PAL16R8*, to select a part number without any prefixes or suffixes. If you use asterisks (*) in the PLD name, ACTIVE-CAD will search the libraries for all PLDs with the matching name. If you type the exact name, ACTIVE-CAD searches only for that name. Once you use this option, ACTIVE-CAD remembers the last PLD that you used and will display its name so that you can load it again without searching.

NOTE: You must add the PLD library to your project (see Project Libraries), because only the libraries selected for the current project are searched for the selected PLDs.

Loading JEDEC file into a PLD

The **Test PLD** window also allows you to select the JEDEC files to be loaded into the selected PLDs. If there are some previously used JEDEC files in the current project resources, they will be displayed in the JEDEC file window. If you want to load additional JEDEC files, use the **Browse** button, which allows you to select a JEDEC file from any disk location and add it to the project resources.

Before you load a PLD, you should decide if you want to load both pin names and test vectors from the JEDEC file. The **Load Pin Names** option in the **Test PLD** window allows you to load the original pin names that have been used as the external PLD pins. This information can be retrieved from the JEDEC file or from the PLC file.

You need to consult your PLD Development Tools documentation for details on whether the pin names have been saved in the JEDEC file or in the PLC file. For example, the PALASM software generates the pin name information in the *.PLC file, and the MACHXL software puts it directly into the JEDEC file.

Loading PLD Test Vectors

The **Load Test Vectors** option in the **Test PLD** window allows you to retrieve the test vector information from the JEDEC files. The JEDEC file format provides test vector information that is typically used for testing PLDs in production. These test vectors are created by the PLD Develop-

ment System when you run the functional simulation, based on the logical equations from the design file. When you load this information into ACTIVE-CAD, the test vectors will be displayed in the **Waveform Viewer**. ACTIVE-CAD loads both input and output test vectors, as shown in Figure 2-8.

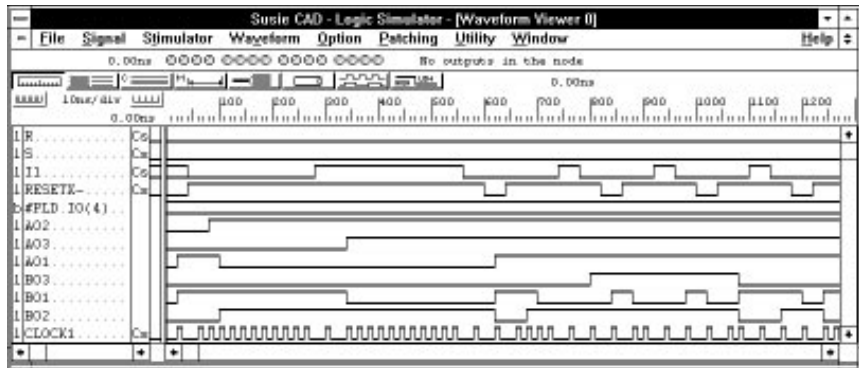


Figure 2-8. Test Vectors Loaded from a JEDEC file.

Since the JEDEC test vectors do not have any timing information, you need to specify the **Test Clock Rate** shown in Figure 2-7 to perform timing simulation. This option requires that you enter the one-half clock period in nanoseconds. For example, if you specify that **Test Clock Rate** = 5ns (period = 10ns), the PLD will be tested with the 100MHz clock. This setting is important if you are using timing simulation, since a fast clock speed can cause a timing violation in the PLD. Always run the timing simulation to make sure that your design will work with the desired clock speed.

Loading Memory Contents

If you have a memory device in your design, you can load it with a hex file. The **Load Memory Chip** option in the **File** menu invokes the **Select Memory Chip** window (Figure 2-9) which allows you to select a memory device. After the memory device has been selected, ACTIVE-CAD displays a listing of available hex files (Figure 2-10).

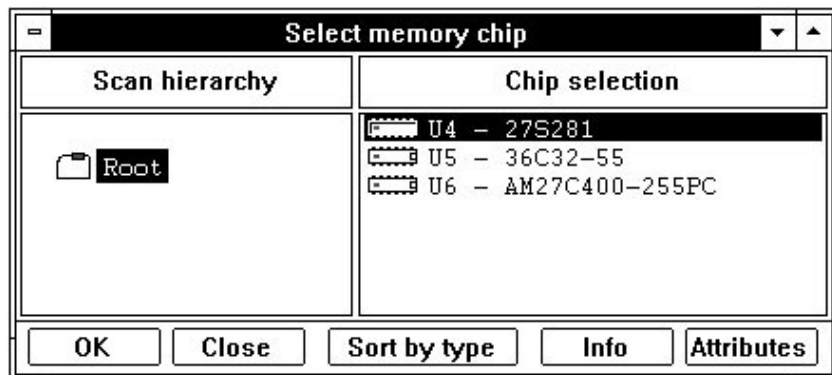


Figure 2-9. List of Memory Components.

Click on the selected hex file and on the **LOAD** button. After loading the hex file into the memory device, the simulation will output the loaded data when a **READ** cycle is simulated. You can save the current memory contents into a hex file using the **Save Memory Chip** option.

NOTE: If you want to load an external hex file, you need to add it to the project resources with the help of the **Resources** option in the **Project Manager** window. You can also add it using the **Browse** button in the **Load Memory Chip** window. All hex files that you save within ACTIVE-CAD are automatically added to the Project Resources.

Loading hex files into hierarchical designs

To load a hex file into memory or a microprocessor select the **Load Memory Chip** option from the simulator **File** menu. ACTIVE-CAD will display the window shown in Figure 2-9 , which displays the design hierarchy. Click on a schematic macro that has some memory devices, and ACTIVE-CAD will list (in the **Chip selection** window) all the memory devices located within that schematic macro. Double-click on the desired memory device and ACTIVE-CAD will instantly display the window shown in Figure 2-10 which lists all available hex files. Select the desired hex file and click on the **Load** button. You can also activate the **Browse** option and load a hex file from other system directories.

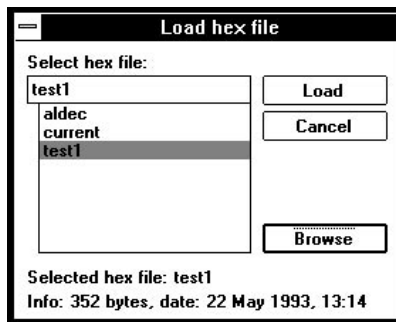


Figure 2-10. Load HEX File Window.

NOTE: When you click on any device that has internal memory, e.g., 8051; It will be displayed as a memory chip in Figure 2-9 so you will be able to load the desired hex file from Figure 2-10.

Memory Range

Memory devices are simulated in ACTIVE-CAD like real hardware. This means that the data can actually be written, stored and read from the memory device by simulating appropriate cycles. You can also view and edit memory contents using the **Memory Editor** option within the **Utilities** menu.

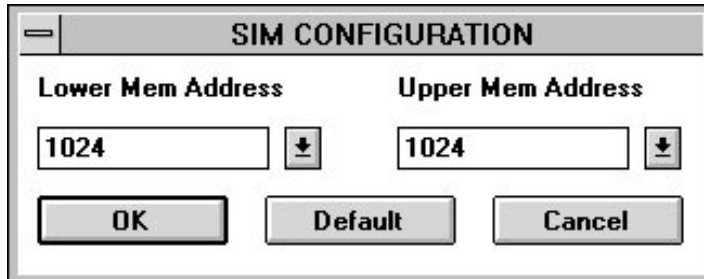


Figure 2-11. Define memory contents range.

The memory contents are stored in your computer's RAM, so if you simulate 1MB memory devices, it can take 1MB of your computers memory to store the entire memory contents. In most cases storing the entire memory contents is not necessary, because even partial code simulation may be sufficient for hardware interface verification.

ACTIVE-CAD allows you to simulate the hex code located at the lowest and the highest memory device addresses. To specify the amount of memory to be simulated (stored by the simulator) click on the **Memory Range** option in the **Options** menu, invoking the **SIM Configuration**

window shown in Figure 2-63. Select the amount of memory in the lower and upper memory addresses to be stored and simulated. The **Lower** memory stores the requested number of bytes, starting from address 0. The **Upper** memory stores the requested number of bytes in the simulated memory device, starting from the highest address.

NOTE: The memory range setup applies to all memory devices in the entire design.


Selecting Signals for Display

Selecting Probes on the Schematic

If you are using the ACTIVE-CAD schematic editor connected on-line to the simulator, you can select the simulator signals directly on the schematic. To do so, activate the **Probe** mode button in the schematic editor and then select the test points that you want to observe or stimulate, such as node names, I/O terminals or device pins. The selected test points will be instantly displayed in the simulator's **Signal** field. If you click on an IC in the schematic editor, a window with the IC pin list will pop up on the screen. Select the desired pins, using the left mouse button, and confirm the selection with the **Add** and **OK** buttons. The selected pins will be copied to the simulator's **Signal** field and the corresponding schematic test points will be marked with probe symbols.

NOTE: To delete probes, click on their schematic square symbols again.

Selecting Signals in the Simulator

To select signal names and IC pins to display in a waveform window, click on the **Select Probe** button  (#6 in Figure 2-2), or select the **Add Signals** option from the **Signal** menu. The **Component Selection** window will appear (Figure 2-12), which will list all parts and signals.

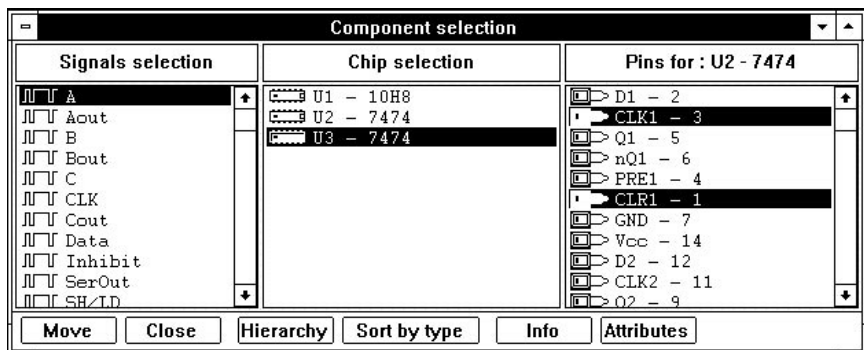


Figure 2-12. Selecting Pins And Signals.

To make your work easier, ACTIVE-CAD offers four (4) ways to add the signals and pins to the simulators display.

Individual pin/signal name transfer:

- ☐ direct dragging; just click on the selected signal or pin name in Figure 2-12 and holding the mouse button down drag it to the Waveform Viewers **Signal** field and then release it.
- ☐ double-click on the selected pin or signal name in Figure 2-12. It is instantly moved into the **Waveform Viewer** window.

Group signal transfer:

- ☐ Click on the first signal/pin name and then click on the remaining ones holding the **{Ctrl}** key down. When all selected signals/pin names are highlighted, click on the Move button. If the signals to be selected are adjacent to each other, click on the first one and then holding down the **{Shift}** key click on the last signal/pin name. When all selected signal/pin names become highlighted, click on the Move button.
- ☐ Position the cursor over the **Signal Selection** or **Pins For** field (Figure 2-12) and click the right mouse button. When ACTIVE-CAD displays local menu (Figure 2-13), click on the **Select All** option. Next, holding down the **{Ctrl}** key, click on the signals that you want to de-select from the group of signals/pins. After the selection has been completed, click on the **Move** button.

NOTE: If you select a signal line in the **Signal** field (highlighted blue) before transferring any signals, then all subsequently transferred signals and pins will be placed in the **Signal** field above that location.

<u>M</u> ove
M <u>ove all</u>
<u>I</u> nsert empty line
<u>S</u> elect all
<u>D</u> eselect all
S <u>e</u> arch for ...
<u>B</u> us mode on/off
F <u>i</u> nd ... in <u>S</u> C
<u>V</u> iew Connections
<u>S</u> how Alias

Figure 2-13. Local Signal Selection menu.


The Menu in Figure 2-13 has the following options options:

- ☐ **Move** - moves previously selected signals
- ☐ **Move All** - moves all listed signals
- ☐ **Insert Empty Line** - inserts an empty line above the selected (blue) signal line in the **Signal** field; empty signal lines cannot be inserted within a group of bus signals
- ☐ **Select All** - selects all listed signals; If the majority of signals need to be transferred, select all of them. Then holding down the **Ctrl** key and clicking the mouse button, deselect the undesired signal lines.
- ☐ **Deselect All** - deselects all selected signal lines
- ☐ **Search for** - allows you to search for a selected signal name in the design netlist
- ☐ **Bus Mode On/Off** - switches the bus display between a single line and discrete lines
- ☐ **Find in SC** - shows the desired signal name on the DDE-linked schematic design
- ☐ **View Connections** - shows the signal connections and logical values

If you click the right mouse button in the **Pins For** field, Figure 2-13 appears to facilitate the transfer of device pins into the **Signal** field.

The **Signals Selection** field displays individual signals and buses. If a bus signal is provided within the loaded netlist, it is displayed as a single bus line. However, after its transfer into the **Signal** field:

- ☐ all bus signals are listed in discrete form

- ☐ the first discrete signal name in the bus is marked with * and under this name the bus will be displayed and stored
- ☐ the following bus members include the + sign indicating a bus membership
- ☐ to convert the discrete bus lines into a single bus line, click on the **BUS** button (#5 in Figure 2-2) 

When the selection of signals has been completed, click the **Close** button to close the **Component Selection** window in Figure 2-12.

Selecting Device Pins

To select device pins, double-click on the desired component in the Chip Selection field (Figure 2-12). ACTIVE-CAD will display a list of pins of the selected device or macro. This list is shown in the right-hand window of Figure 2-12. The selection of device/macro pin names to display is similar to the selection of signal names (double-clicking, dragging, **Move** button, the right mouse button menus).


The **Pins For** field may display both the external and internal device pins. If the external pins are transferred into the **Signals** field, they can be overridden by the designer with local test vectors. However, the device internal test points can only be used for display of signal waveforms and cannot be overridden.

Chip Selection Field

The **Chip Selection** field in Figure 2-12 lists all the components at the selected hierarchical level. The gray devices and macros are active electronically. The white ones are either disabled by the selective simulation option or they have no model and will not be simulated.

NOTE: When simulating a design, check that all devices and macros in the field are gray. If any of them is white, it is a warning that an incomplete simulation is taking place because the white devices have no models and produce High_Z outputs.

Simulating hierarchical designs

ACTIVE-CAD can simulate hierarchical designs, which are easier to manage than flat designs. To access various hierarchical levels, select the **Component Selection** for Waveform Viewer window by clicking on the **Select Probes**  button. Note that the right-hand column displays the design hierarchy structure (see Figure 2-14). When you click on the desired hierarchy level, the simulator will display the components

and signals present at that level. Following this, you can select the desired signals and pins to display.

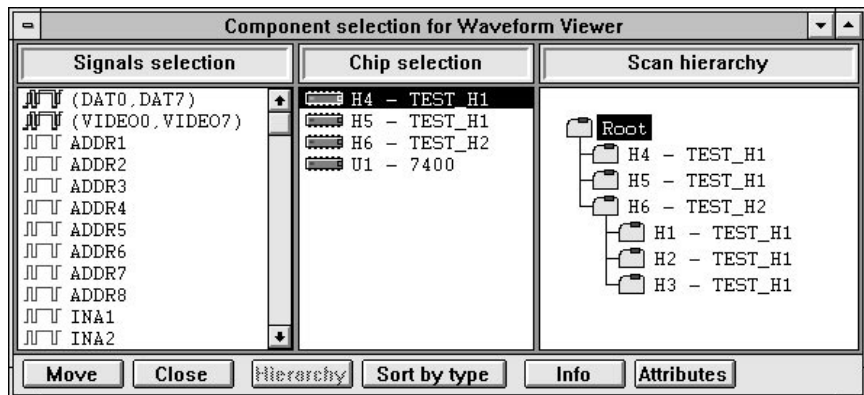


Figure 2-14. Selecting probes in selected hierarchy.

A design may have the same signal and device names at various hierarchical levels. In order to distinguish them by hierarchical level, highlight the signal and select the **Hierarchy** option from the **Signal** menu. This displays the hierarchical level of the selected signal.

If you double-click on the desired component, ACTIVE-CAD displays all its I/O pins. To display the desired component pins on the simulator screen, select them either with the mouse button (double-click) or by using the **Move** button. This copies the selected pin(s) to the simulator display screen. The **Signals Selection** field displays only the node names. It does not display any hierarchical terminals, except for the root level.

This procedure for selecting test points can be applied regardless of device hierarchical location.

NOTE: Since terminals of hierarchical schematic sheets are duplicated as symbol pins at the next hierarchical level, you must select them at that level for display. You will not find them in the **Signal Selection** field.

Defining Buses

You can define any set of signals as a bus and simulate it accordingly. To create a bus, follow these steps:

1. Using methods described earlier, copy the member signals from the Component Selection for Waveform Viewer window to the Signals section of the waveform Viewer. You may copy the signals in the or-

der in which you want them to appear in the bus, or you may sort them in the next step.

2. You must make sure that the member signals are listed together and are sorted in the order in which you want them to appear in the bus. The signal listed first from the top will be the least significant bit, and will represent bit 0 of the hex bus. You may move signals in the list by dragging them to their new locations (see the next section).
3. Select (highlight) the bus members. Click on the first member and then click on the last member of the bus while holding down the **Shift** key. You can also click on each bus member while holding down the **Ctrl** key .
4. Select the **Create** option from the **Signal/Bus** menu. The selected signals are instantly converted into a bus. Use the **Bus** button (#5 in Figure 2-2) to toggle the bus display between discrete lines and condensed (hexadecimal, octal, etc.) display.

Clicking on a new signal name, while the **Ctrl** key is depressed, selects the new signal but does not deselect the previously selected ones.

The bus operations menu is shown in Figure 2-15:

- ☐ **Create** - converts selected signal lines into a new bus; the first signal name in the bus automatically determines the name of the bus
- ☐ **Destroy** - converts the selected bus into a set of unrelated discrete lines
- ☐ **Bus Direction** - reverses the order of the bus lines, e.g.. swaps the signals order from DATA0-DATA15 into DATA15 - DATA0
- ☐ **Bus Name** - allows you to assign a special name to the selected bus
- ☐ **Display Binary** - converts the bus display into a binary format
- ☐ **Display octal** - displays the bus in octal format
- ☐ **Display decimal** - displays the bus in decimal format
- ☐ **Display hexadecimal** - displays the bus in hexadecimal format

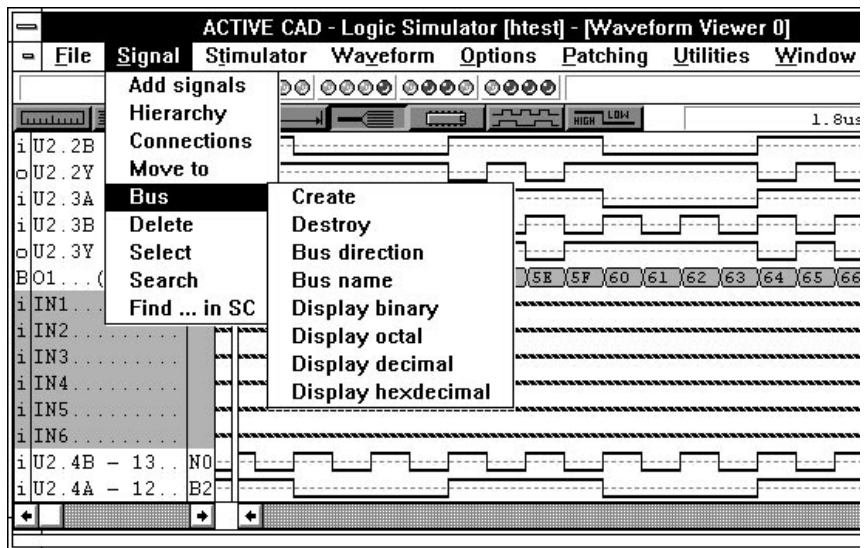



Figure 2-15. Defining a bus.

To toggle between discrete signals and the hex bus representation, click on the bus button  in the **Waveform Viewer** window tool bar (#5 in Figure 2-2). To permanently deselect a bus, click on a bus and select the **Destroy** bus option from the **Bus** submenu shown in Figure 2-15. The entire bus will instantly be converted back to discrete signal lines.

NOTE: All signal lines drawn as buses in the ACTIVE-CAD Schematic Editor or loaded as buses within the design netlist will automatically be displayed in the hex format.

Moving Signals

From time to time, you may need to rearrange signals on the screen. To rearrange the order of signals, click on the desired signal with the left mouse button and when a waveform (_ _ _) icon appears, drag it to a new location where the signal should be inserted. The simulator instantly moves the signal name and the associated waveform to the new location and the signal previously residing at this location is pushed down by one location.

NOTE: If you drag a signal outside the **Signal** window, it will be deleted from the waveform window.

View Connections

The **Connections** feature, selectable from the **Signal** menu, allows you to review the pin connections and output signal logical states that are generated by IC models. This feature is particularly important when you are trouble-shooting IC models at the system level or when you are tracing netlists generated by external schematic editors.

Selection Process

To view the connections, follow these steps:

- ☐ Select signals for display in the simulator window
- ☐ Click on the desired signal to turn it blue
- ☐ Select the **Connections** option from the **Signal** menu in Figure 2-15.

ACTIVE-CAD will display the **Connections** window shown in Figure 2-39, which lists all node connections for the selected pin/signal line. The **Connections** window also displays logical signal states of all pins and signals after you click on the **States** button located at the bottom of the window. You can view only one node at a time. If you have selected more than one signal line, ACTIVE-CAD will display only the node connections for the top-most selected (blue) signal line.

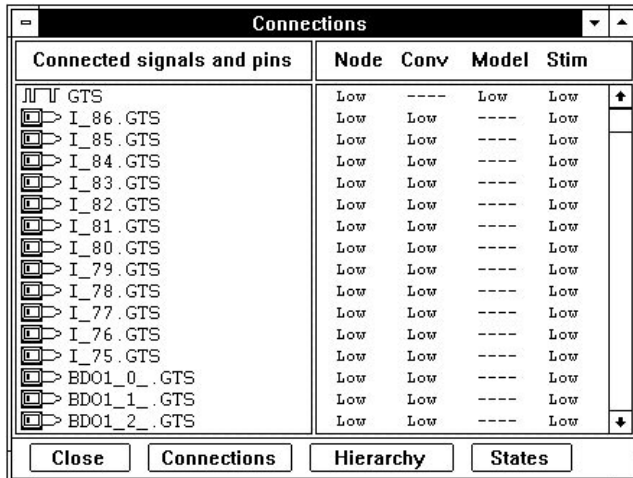


Figure 2-16. View Connections Window.

Operations

The left-hand field of Figure 2-12 displays all device pins in the selected node. If you double-click on a pin, all other pins of that device will also be displayed. Double-clicking on any of these pins will select its connec-

tivity node for display. You can continue this process, until you trace the signal through all the devices in its path. Practice the signal tracing on a flat schematic to understand the signal path tracing procedure. Following this, you should practice signal tracing in hierarchical designs.

Clicking on the **Hierarchy** button in Figure 2-16 will activate the hierarchy display, which shows the hierarchical location of the selected pin or signal node. The **States** button allows you to toggle between the expanded and abbreviated **Connected signals and pins** display fields. The expanded field allows you to view the logical states of all the selected pins in the node.

The expanded right-hand field in Figure 2-39 has four fields: **Node**, **Conv(ersion)**, **Model** and **Stim(ulator)**. The **Node** field shows the resulting signal state by combining all signals produced in the node. For example, if the node is a bus, then it shows the highest strength signal that prevails. If there is a signal conflict in the selected node, this field will show Unknown or X signal state.

The **Conv(ersion)** field shows how the model converts its input signal. For example, if you apply on the input to the TTL gate a supply voltage logical state (SU_V), ACTIVE-CAD will convert it internally to a logical HIGH state and this value will be shown in the **Conv(ersion)** field.

The **Model** field shows the state on the models output. For example, if there are a few output pins (sources) in a node, this field will show the actual signal state as produced by the model itself, independent of the resulting **Node** signal state.

The **Stim(ulator)** field shows the logical state of an external stimulator applied to the selected pin. After Power-On, all external stimulators are set to the LOW logical state. You need to simulate a single step to restore the **Stimulator** signals to their actual logical states. If no **Stimulator** signal is applied at the given pin, the field shows blanks. Remember that the stimulators override the devices outputs only under certain circumstances (e.g., if the node is in the stimulator **override mode**, or if you apply a stimulator to a gate with a weak output signal).

Test Vectors

A test vector is the state of all displayed signals at a particular time instance. It is a vertical slice of the signal waveforms at any given time. Test vectors are used as input signals that stimulate the design and produce simulation results. A test vector file for manufacturing test applica-

tions may also include some output signals that are used for comparison with the hardware-produced signals.

Test vectors can be entered into ACTIVE-CAD in the following ways:

- ☐ Permanently assigned
- ☐ Keyboard key controlled
- ☐ Binary counter driven
- ☐ Formula stimulator
- ☐ Asynchronous clock editor
- ☐ Graphical waveform editor
- ☐ Advanced Test Vector Editor (Active CAD)
- ☐ Test Vector Macro Editor (ACTIVE-CAD)
- ☐ External test vector files

Test vectors can be combined from many sources and exported or saved to disk (binary or ASCII). You can also output test vectors to printers and plotters in graphical form.

ACTIVE-CAD Test Vector Generator Panel

There are two basic approaches to stimulating designs:

- ☐ Using signal waveforms; each signal waveform is specified from time t_0 till end of simulation
- ☐ Using test vectors; they describe ALL signals at discrete time instances when at least one of the signals changes its logical state

Signal waveforms have been very popular with hardware designers who have been accustomed to setting up signal generator channels that control each input signal line. To accommodate the complexity of ASIC designs and test equipment and the limitations of batch simulators, the concept of test vectors has been developed.

Today, most designs are expressed in test vector files. If you simulate a design with a set of signal waveforms and then save it as an ASCII test vector file (**Save ASCII Test Vectors** in the **File** menu), you can use these test vector files with ASIC and board testers.

Developing signal waveforms is generally easier than developing test vector files, particularly for incremental design analysis. Figure 2-17 shows a test vector generator panel that pops up when you click on the Stimulus button or when you choose the Add Stimulators option from the Stimulator menu.

ACTIVE-CAD provides for five (5) modes of signal waveform generation:

- ☐ **Keyboard** - any key (A-Z) can be assigned to any signal line and toggled while the simulation is in progress. There are altogether 26 keys which can be used for direct hardware control.
- ☐ **Bc** - these 16 green lamps represent 16-bit binary counter outputs which can be used as basic design stimulus or clocks. Their advantage is great simplicity of use. They are limited to a 50% duty cycle.
- ☐ **NBc** - these are inverted Bc counter outputs
- ☐ **Form(ula)** - it allows to define any signal waveform through nested expressions. It is a simple and effective way to develop the most complex design stimulus signals.
- ☐ **Clocks** - this is a derivative of the Formula option which assures repeated execution of the basic signal waveform formula

You can assign a stimulus signal to signals or pins in the following ways:

- ☐ Click on the stimulus signal and drag it over the selected signal
- ☐ Click on the selected signal and when it turns blue, click on the stimulus signal in Figure 2-17.

In addition to the five signal generation fields there are several control buttons which facilitate speedy implementation and disconnection of the applied signal waveforms:

- ☐ **Delete** - deletes the stimulator assigned to a signal name
- ☐ **EN** - enables the previously disabled signal stimulator
- ☐ **DS** - disables the assigned stimulator without deleting it
- ☐ **CC** - disables the override signal function
- ☐ **OV** - when assigned to an output pin, it overrides the chips output
- ☐ **Mode:CC** - enables chip outputs to control their nodes
- ☐ **CS** - forces an existing signal waveform to act as an input signal

To assign the control button, use the following procedure:

- ☐ Assign stimulus signals to signals and pins
- ☐ Click on a selected signal or a pin to turn it blue
- ☐ Click on the selected button.

The button activity will effect the color of the stimulus signal:

- ☐ **EN** button - turns the stimulus signal bright

- ☐ **DS** button - turns a bright stimulus signal into a gray one
- ☐ **CC** button - turns red stimulus signal into a gray one
- ☐ **OV** button - turns black stimulus signal into a red one
- ☐ **Mode:CC** - disables all override signal stimulus
- ☐ **CS** button - freezes the existing test vector and overrides simulation results on device input pins

Binary Counter

The ACTIVE-CAD simulator has a built-in 16-bit, software-driven binary counter, which can be used as a source for test vector generation. Its bits, designated B0-BF, are displayed at the top part of the simulator window as a set of 16 LED lamps. You can change the clock of this binary counter by selecting the **Clock Settings** option within the **Options** menu, and following the window prompts.

You can toggle the initial state of each bit of the binary counter by placing the cursor over the desired bit in the counter display (Figure 2-17) and pressing the mouse button. In response, the color of the selected LED will change to the opposite logical state.

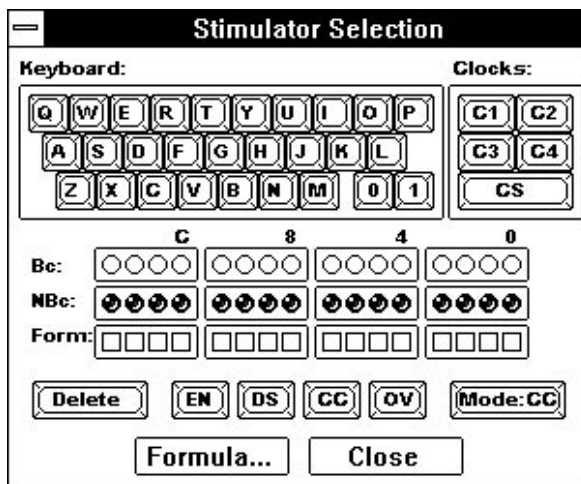



Figure 2-17. Stimulus window.

Keyboard Key Controlled Signals

Since ACTIVE-CAD is a real-time logic simulator, you can assign any keyboard key **a** through **z** to a selected test point and toggle it while the simulation is in progress. To assign a keyboard key to a selected test

point, select a signal in the **Signal** field and invoke the **Stimulus** window (Figure 2-17) by pressing the **Stimulus** button  or clicking on the Add Signals option in the Signals menu. Next, click on one of the keyboard keys shown in Figure 2-17, to assign it to the selected test point. In response, the selected character will be displayed next to the signal name in the **Stimulus** column (#17 in Figure 2-2).

NOTE: Since ACTIVE-CAD does not know which logical state should be assigned to the selected signal line, it assigns by default the high impedance state. To assign High or Low logical state, you must toggle the assigned keyboard key till the desired logical level appears.

NOTE: The keyboard keys can toggle only between logical 1" and 0".

Assigning permanent logical levels

The keyboard keys can only provide High and Low logical states. If you must feed any other logical state or if you want to permanently assign any logical state, follow this procedure:

- ☐ Assign a keyboard key to a selected signal line
- ☐ Select the signal line again (turns blue)
- ☐ Bring up the **Test Vector State Selection** window (Figure 2-18). Next, click on the selected logical state button in Figure 2-18. The selected logical state is assigned to the signal line and displayed next to the assigned keyboard key
- ☐ The selected logical state will be permanently assigned to the selected signal line and the associated keyboard key will no longer toggle the signal line. To remove this assignment, delete the previously assigned keyboard key.

Graphical Waveform Editing

Since ACTIVE-CAD is a real-time simulator, test vectors can be created directly while the simulation is in progress. You can force any signal waveform on the device output and emulate the desired operation. To account for any possible device output state, the ACTIVE-CAD test vector editor allows you to choose any one of the fifteen (15) different signal states.

To create a custom signal waveform that will be used as a design stimulus or an input test vector, perform the following operations:

1. Click on the **Edit** option within the **Waveform** menu. This displays the **Test Vector State Selection** window with 15 logical state buttons (Figure 2-18).

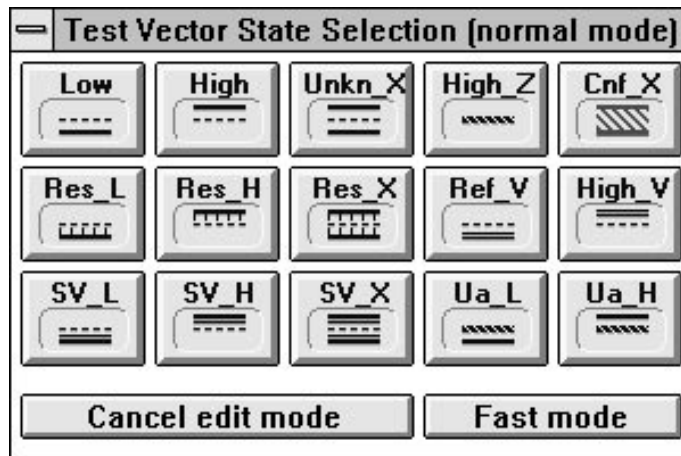


Figure 2-18. State selection window.

2. Move the editing cursor to the desired signal row and column (time) in the waveform window and click with the left mouse button. A blue vertical cursor appears in the waveform window (#10 in Figure 2-2) as a time reference. Note that the edited signal name is now highlighted in green.
3. Click on the desired logical state in the **Test Vector State Selection** window. This will create a waveform with the desired state between the selected time location and the closest previous signal transition on the selected signal. If there was no prior signal transition, the waveform will start from the time 0ns.
4. Following this procedure, you can create any number of signal changes by selecting the change location with the blue vertical cursor and then selecting the appropriate signal logical states from the **Test Vector State Selection** window (Figure 2-18).
5. To exit the editing mode, click on the **Cancel Edit Mode** button.

NOTE: The **Cs** (Custom Signal) marker is automatically inserted for each edited signal waveform. This marker directs ACTIVE-CAD to treat the graphical waveform as an input signal. If the **Cs** marker is assigned to an output signal, it must be set to the override mode (red **OV** button) to override the device-generated output signal.

Since the signals produced by the Waveform Editor may change the existing waveforms, they are drawn in green for better differentiation.

Formula Waveform Editor

Invoked by selecting the **Formula/Edit** option from the **Waveform** menu, the **Formula Editor** (Figure 2-19) is a tool for quick test vector development. Using the items listed in Table 2-1, define the signal waveform by entering the formula into the editor shown in Figure 2-19.

Examples:

H40L10	High for 40 ns then Low for 10 ns
(H40L10)20	Same as above but repeated 20 times
H4usL1us	High for 4 us (microseconds) then Low for 1 us
((H10L10)20x30)10	High for 10 ns then Low for 10 ns, repeated 20 times, after that Unknown (X) for 30 ns. Repeat the entire signal waveform segment 10 times
([2]40[A0]55)10	02hex for 40ns, then A0hex for 55ns. Repeat 10 times

Table 2-1. Symbols Used in Formula Editor.

Symbol	Description
H, L	high and low logic levels (1 and 0)
X	unknown (high or low)
Z	high impedance
0..9	numbers used for defining duration and repetitions
()	parentheses for selecting sub-expressions
ps, ns, us, ms	time unit definition for duration arguments, default is ns (nanoseconds)
[]	brackets for defining hex bus value

NOTE: Waveforms generated by the Formula Waveform Editor cannot be distinguished from any other waveforms. Once placed on the screen, you can modify these waveforms using the waveform editor.

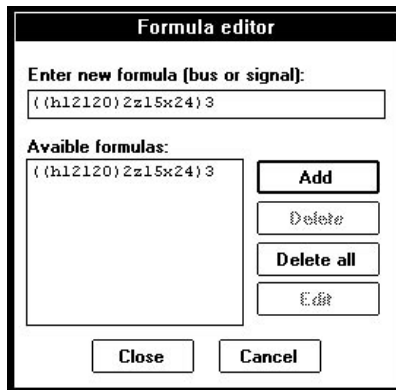


Figure 2-19. Waveform Editor

After you have edited the waveform equation or formula, click on the **Add** button and close the window. Next, place the cursor in the waveform area, and press the left mouse button to bring the blue vertical cursor to the screen location where you want to insert the waveform formula. The background of the selected signal name will change to green. Use the **Formula/Insert** option from the **Waveform** menu to insert the edited waveform at the desired location.

If you select the **Formula Replace** option, a new waveform will replace the existing signal waveform.

NOTE: The Waveform formula calculates the signal logical states starting from the point of its insertion.

Formula-based Stimulators

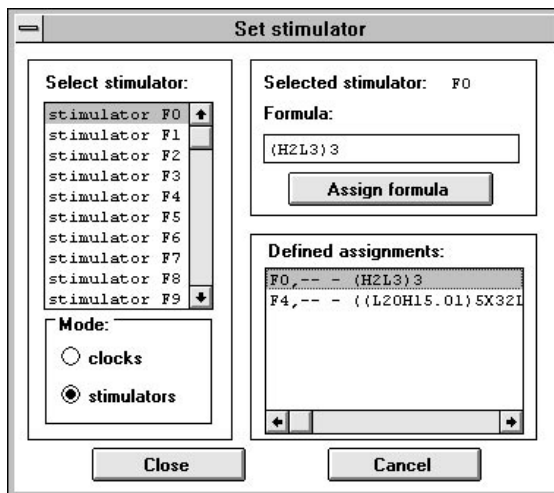


Figure 2-20. Defining Formula Stimulator.

You can assign a formula to a signal using the **Stimulator Selection** window in Figure 2-17. The square lights in the **Form:** field represent 16 formula-based stimulators that can be defined by the user.

To define a formula stimulator, click on the **Formula** button in **Stimulator Selection** window. The dialog box in Figure 2-20 appears.

Double click on one of the formula stimulators in the list on the left. The formula name will be displayed as the **Selected stimulator** in the right-hand side window (e.g., F4). Type in the test vector formula using Table 2-1 and press the **Assign formula** button. The new formula is then displayed in the **Defined Assignments** field shown in Figure 2-20.

The assigned formula will always be calculated from the simulation origin (time 0ns) and will be displayed on the screen as a signal waveform when you simulate. You do not need to redefine a test vector formula for resimulation.

NOTE: If you start the simulation at the time t_n , ACTIVE-CAD will calculate the signal from its formula, starting at time t_0 and display the signal starting at time t_n .

To change an existing named waveform formula, first select it from the formula list, edit it and then click on the **Assign Formula** button.

Assigning Formula Stimulators


To assign a formula stimulator to a signal, click on one of the formula square buttons (light 0-F), drag it over the desired signal and then release the button. The **Fx** marker should be displayed next to the signal in the Stimulator column. Another way to assign the formula stimulator is to first click on the signal or signals so that they are highlighted, and then click on one of the formula buttons. The selected formula F0-FF will be instantly assigned to the selected signal (s).

Asynchronous Clocks

ACTIVE-CAD allows you to have up to four (4) asynchronous clocks which behave like four independent crystal clocks. These clocks can have a resolution of 10 picoseconds each and are completely independent of each other. Moreover, these clocks can be complex waveforms, including pulse bursts, etc.

The **CLOCK** waveform is a test vector that is automatically repeated as the simulation progresses. To create a clock test vector, select the **Clocks** option in the **Mode** field of Figure 2-20. (see *Formula-based Stimulators* section on how to open this window). Then select one of the clocks from the **Select stimulator** section. Next, enter the clock formula according to the

same rules as any other signal formula, and press the **Assign formula** button.

To assign an asynchronous clock, to a signal line, click on a signal name and invoke the **Stimulus** window by pressing the  button. Then click on one of the **Clock** buttons (C1..C4) displayed in the **Stimulus** window. As a confirmation, ACTIVE-CAD displays the selected clock symbol next to the signal name. The **Clock** buttons (C1...C4) are bright when they have some formulas assigned to them. Otherwise, they remain gray or inactive.

NOTE: You can use any formula signal as a clock. ACTIVE-CAD provides the clock signals in explicit form to underscore the periodical nature of these signals.

ASCII Test Vector Files

You can load ASCII test vectors while you simulate. These ASCII test vectors can be generated during simulation, created using ACTIVE-CAD editor or generated by other test vector editing software that produces ACTIVE-CAD compatible formats.

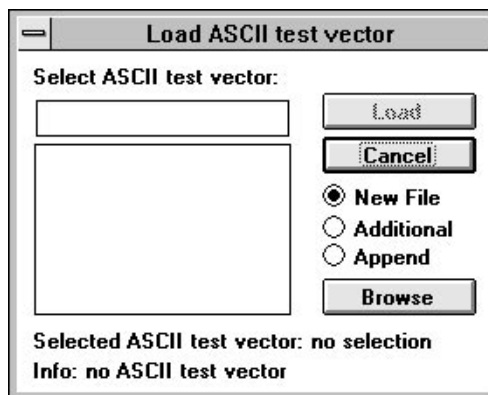


Figure 2-21. Loading ASCII test vectors.

To load an ASCII test vector file, select the **Load ASCII Test Vectors** option from the **File** menu. ACTIVE-CAD will display a list of available ASCII files and gives you three loading options:

- ☐ **New File;** overrides the currently present test vectors
- ☐ **Additional;** loads additional signal waveforms from time 0

- ☐ **Append;** loads additional signals to the right of the current blue cursor location

Click on the selected loading option and test vector file. Clicking on the **Load** button loads the ASCII test vector file. For details on the ASCII test vector file format refer to Appendix A.

Stimulator Override

External signal waveforms applied to a design are subject to the following rules:

- ☐ When applied to a device input pin, the signal waveform always controls it unconditionally. But it does not effect any other device pin in the same signal node.
- ☐ When applied to a device output pin in a non-override mode, it controls the entire node only if the output pin produces a weak signal. If the device output pin produces a strong signal, the external signal waveform has no effect on the signals in the node.
- ☐ When the external signal is set to the override mode, it overrides any device output pin, independent of its strength. The signal override process carries down to the lowest hierarchical level.

To set a signal waveform to the override mode, select it in the **Signal** field and then click on the **OV** button in Figure 2-17. To disable or delete the overriding signal, select the signal in the **Signal** field and then click on the **DS** (disable) or **Delete** button, respectively, in Figure 2-17.

Each signal node is by default defined as **Chip Controlled**. This means that if any of the outputs in the node is active (has a state other than high impedance), then the stimulator is overridden by that output unless the stimulator is in the **Override** mode. To switch back to the **Chip Controlled** mode, click on the desired signal line and then on the **CC** (Chip Controlled) button in in Figure 2-17.

Waveform Display

Selecting Waveform Scale

ACTIVE-CAD displays signal waveforms in a user-selectable scale. The current scale setting is displayed right above the signal names and can be different for each **Waveform Viewer** window that you open.

You can change the waveform scale by activating the **Zoom-In** and **Zoom-Out** buttons, located above the signal names (#15 in Figure 2-2). To change the scale, click on the zoom-in or zoom-out scale buttons which select the next higher or lower predefined display scale, respectively.

If you want to view a selected area of the waveform, click on the waveform ruler at the beginning of the desired waveform area and drag the cursor while holding the mouse button down. A blue line appears over the ruler and follows the cursor. Drag the cursor to the end of the waveform area to be expanded, and release the mouse button. The time segment defined by the blue stripe is expanded to fill the entire **Waveform Viewer** window.

By changing the scale, you can instantly zoom in on critical waveform areas and observe the precise time delays between signal transitions.

Scale Resolution

ACTIVE-CAD is an event-driven simulator. Thus, its accuracy is independent of the selected waveform display scale. The simulator stores the signal events with a 10 picosecond resolution. So no matter what scale you select for display, simulation results will always have the same accuracy. You can prove it by making measurements between the same two signals at different scale resolutions; These measurements will always be identical, independent of the display scale.

The simulation precision is set by default to 10 picoseconds. However, it can be changed to any value between 10 picoseconds and 1 millisecond by using the **Simulation Precision** option in the **Options** menu. Since the ACTIVE-CAD measurement cursor snaps to the signal transitions, the simulation precision determines the snapping accuracy of this cursor.

Running A Simulation

Short and Long Steps

The ACTIVE-CAD logic simulator allows you to simulate designs in short and long steps. To simulate a short step, click on the **Short Step** button, located at the bottom of the simulator toolbox (#7 in Figure 2-1). Similarly, to simulate a long step, click on the **Long Step** button (#8 in Figure 2-1).

The simulation time of the **Short** and **Long** steps is specified in the **Step** window (Figure 2-1) that can be invoked by selecting the **Simulation Step** option in the **Utilities** menu. To change these settings, click on the arrows in the **Short Step** and **Long Step** fields and select the desired step value from the list, or directly edit the step value by clicking on the current value and entering a new value. You can use decimal numbers, decimal point and time units, e.g., 16.34 ms. The default time unit is a nanosecond.

Initialization of the design

Simulation is designed to verify accurately the behavior of logical circuits, so that you can be sure that they will operate properly after they are manufactured. There are, however, some limitations on the exact emulation of the actual hardware operation. For example, most of sequential devices have undetermined logical states on the outputs when they are powered up (a CMOS flip-flop has either high or low logical state when you turn the power on). The simulators can emulate this behavior by selecting at random a high or low state at the beginning of the simulation. This approach, however, is very inconvenient, because each simulation run would result in different data, and you could never be sure that all combinations of the initial device conditions have been tested.

For that reason, a new logical state was introduced into the simulator. It is called the Unknown state and it means that either a High or a Low output signal is present. However, the simulator does not know which logic state is actually asserted. Using this unknown signal state, the simulator will determine if your design is sensitive to random initial states or if the design will reset itself after a few clock cycles to a known initial state. The Unknown state is also generated when the setup or hold times are insufficient for proper device setup and when the preset, reset, load, clock and other signals do not meet their timing specifications or have incorrect time relationships.

Since proper handling of unknown signal conditions is of paramount importance for reliable circuit design, you need to review the comments listed below.

If your design generates unknown output signals, you need to trace the source of the unknown states, which can be:

- ☐ unknown signals at the input of sequential models
- ☐ timing violations; (when you simulate the design in the timing mode)
- ☐ bus conflicts

- ☐ non-initialized sequential devices
- ☐ feedback loops that include sequential components

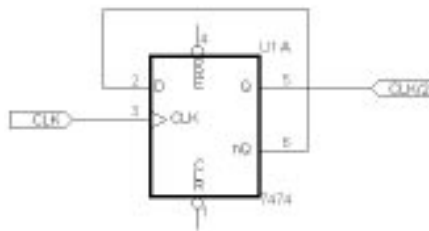


Figure 2-22. Unresolved initialization circuit.

After Power-on, the initial state of the flip-flop in Figure 2-22 is unknown, and so is its D data input. Because of the feedback loop from the nQ output to the D input the flip-flop will always remain in the Unknown state.

To resolve this problem, you can apply a clear (CLR) signal to set its Q outputs to the low logical state. The resulting high logical level on the nQ output will force high on the flip-flop input, and the circuit will start simulating correctly.

If your design has some sequential circuits like flip-flops, registers, counters, etc., but does not have a set or reset line, you can use either the **Power On Settings** (Options menu) or **Selective Preset** (Utilities menu) options:

- ☐ The **Power On Setting** allows you to specify the initial state of all sequential devices in the design.
- ☐ **Selective Preset** allows you to force device inputs and outputs to a specified logical state.

Selective preset

ACTIVE-CAD allows you to preset any signal or device pin to any logical state. Design preset is a two step operation. First, you need to define the preset conditions and then execute (force) these setup conditions. You can edit (define) the preset settings for the entire design and store them in a text file on the disk, and then load this initial conditions file to force the desired design status in various simulation tests.

The preset conditions can be executed at any time during the simulation. The preset option can be used to resolve initialization problems or to test

some design situation that is difficult to generate or takes a very long simulation time to create. For example, if you simulate a 64 bit counter, and you want to simulate the end-of-count sequence, you would normally have to simulate 18,446,744,073,000,000,000 cycles. Instead, you can preset the counter to FFFFFFF0h and simulate only a few cycles to get the same result.

Note: Since the preset signal conditions are forced by the simulator and do not change the IC model internal conditions, the preset signals can sometimes be overridden by the signals generated by the IC model outputs.

Selecting Preset Conditions

To define the preset condition for selected signals and device pins, invoke the **Selective Preset** option from the **Utilities** menu. First, select the **Add** button in the **Selective Preset** window. Next, copy signals and pins for preset conditions from the **Component Selection for Selective Preset** window into the **Selective Preset** window and close the component selection window.

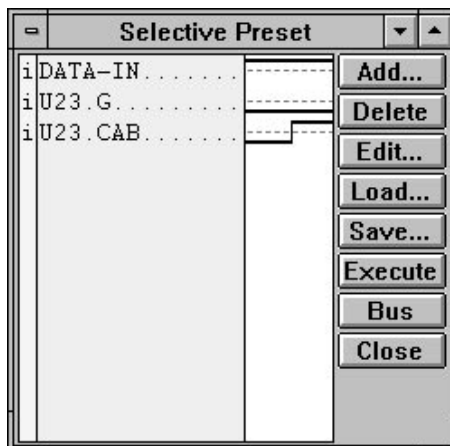


Figure 2-23. Selective Preset Window.

To define the preset logical states on the selected signals and pins, click on the **Edit** button. ACTIVE-CAD will display the **Set Selective Preset** window (Figure 2-24) which shows all logical states available for preset. Select a signal(s) in the **Selective Preset** window, so it is highlighted blue, and then click on the desired logical state in the **Set Selective Preset** window. This logical state will now appear next to the signal name (in the **Selective Preset** window in Figure 2-23).

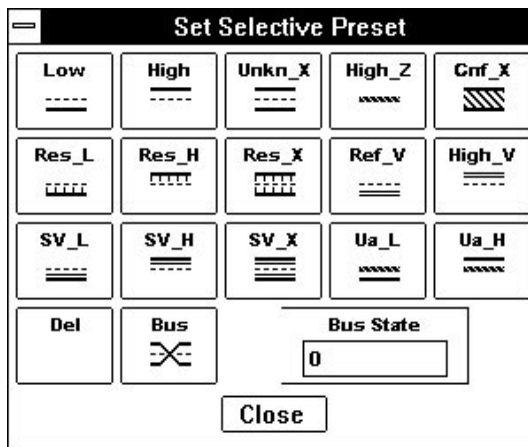


Figure 2-24. Selective Preset Window.

Using the **Save** and **Load** buttons in the **Selective Preset** window (Figure 2-23) you can save and load the preset condition into a text file. There can be multiple preset files available for the same design which you can load into the design when needed.

Forcing Preset Conditions

The **Execute** button executes the conditions specified in the **Selective Preset** window. You can execute or load the new preset conditions as many times as needed during the simulation.

If you have buses defined on the schematic, you will be able to preset the entire bus state using the hex values. To display buses in the **Selective Preset** window in the hex mode, press the **Bus** button.

NOTE: Load preset file operation loads signals into the **Selective Preset** window. You must click on the **Execute** button to force these presets.

Power On


You can reset the entire design by activating the **Power On** button, which is located in the simulation toolbox (#17 in Figure 2-1).

The power-on sequence is as follows:

- ❑ First, all IC models are internally reset to their power-on states. Typically all sequential circuits are reset to the X_Unknown logical state.

- ☐ Next, the outputs are set to the logical state specified in the **Models** field of Figure 2-58
- ☐ Following this, the selective preset is being executed, if the **Execute Preset** box has been checked in Figure 2-58
- ☐ Some IC models have a **Global Reset** command included in their source code. If the **Global Reset** box is checked in Figure 2-58, then these models will be set to their global reset states.
- ☐ Some IC models have hardware controlled resets, such as the **GSR** signal line in the XILINX devices. These reset commands will be executed last, and the model will remain in that condition when the power-on process is completed.

Power On is automatically executed at the beginning of a simulation and every time you click on the **POWER** button. However, it is completed after the first simulation step. So if you perform Global Reset right after Power On, but before the simulation step, it will not be executed.

Clicking on the **Power On** button moves the simulation cursor to the beginning of the waveform display (0ns), but does not delete the signal waveforms. To delete these waveforms, you need to select the **Signal** menu and click on the **All Waveforms** option within the **Delete** sub-menu or click on the **Delete Waveforms** button .

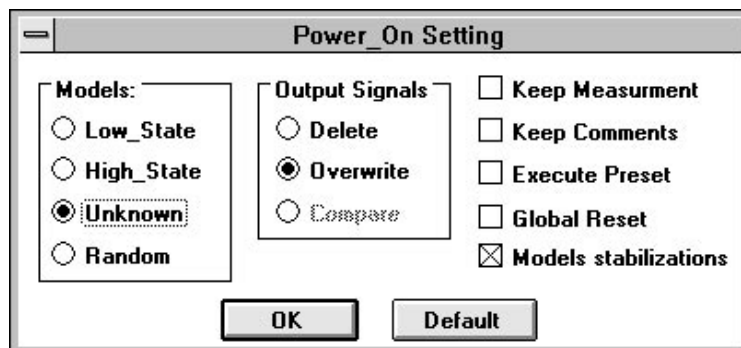


Figure 2-25. Defining Power On Settings.

The **Models** field in Figure 2-25 has check boxes for controlling the models outputs:

- ☐ **Low_State** sets all model outputs to the Low logical state
- ☐ **High_State** sets all model outputs to the High logical state
- ☐ **Unknown** sets all model outputs to the Unkn_X logical state; This is the most popular setup for design analysis.

- ☐ **Random** sets all model outputs to a random logical state

The **Output Signals** field in Figure 2-25 controls overwriting of the old simulation results and has the following check boxes:

- ☐ **Delete** deletes the output signal waveforms after the first simulation step
- ☐ **Overwrite** displays the old simulation results after the Power On; New simulation data overwrites the old one.
- ☐ **Compare** keeps the previous simulation data for reference; The new simulation waveforms change color in the areas which differ from the previous simulation results.

The **Power_On Setting** widow in Figure 2-25 has additional miscellaneous simulation and display check boxes:

- ☐ **Keep Measurement** will keep measurements from the previous simulation run, even if they are no longer valid
- ☐ **Keep Comments** will keep the comments from the previous simulation
- ☐ **Execute Preset** will preset device pins displayed in the **Selective Preset** window
- ☐ **Global Reset** will activate models with the global reset feature
- ☐ **Models Stabilization** forces termination if 10,000 events have been detected during Power On

The **Power_On** Setting window allows you to control the simulation starting conditions and simulation reruns.

Simulation Modes

As the design analysis progresses, your needs may change. From a simple functional simulation, you may switch to unit propagation delay or timing analysis. The ACTIVE-CAD toolbox in Figure 2-1 has a simulation **Mode** button (#13) that is used for the simulation mode selection. Clicking on this button displays a new simulation symbol and switches the simulator from one simulation mode to another.

- ☐ **FM** is a functional mode with 0 propagation delays
- ☐ **TM** is a high resolution timing mode
- ☐ **GL** is a functional simulator with an adjustable unit delay
- ☐ **UN** is a factory-set unit delay functional simulator

NOTE: Since ACTIVE-CAD is a real-time interactive simulator, the change to a new simulation mode takes instant effect.

Functional Simulation

Functional simulation is simulation with zero propagation delays. It means that outputs are changing instantaneously in response to new device input conditions.

To select functional simulation, select the **FN** option from the simulation toolbox, (#13 in Figure 2-1). The simulation mode button is in the center of this toolbox. The functional mode takes effect instantly, without any need for compilation. All newly generated waveforms will display functional behavior of all devices.

When the functional mode is selected, all device output signals change at the same time as the associated input signals, meaning that designs with feedback signals can cause oscillations. Though the simulator stops processing these designs after exceeding a predefined number of oscillations (10,000 events), nevertheless the designs can be successfully simulated in the **Timing (TM)**, **Unit Delay (UN)** and **Glitch (GL)** modes.

Unit Delay Simulation

To select the **Unit Delay** mode, click on the **Mode** button in the simulation (#13 in Figure 2-1) toolbox until it displays **UN**. The **Unit Delay (UN)** simulation mode is based on unit propagation delays, which means that all components have the same delays from input to output. The value of this standard delay is factory-set to be equal to the simulators current resolution.

The **Unit Delay** mode is used for functional verification of designs that have oscillating feedbacks, and cannot be simulated in the functional mode. No timing errors are reported in the UN mode. However, any glitches will be reported and displayed in the waveform window, providing that the associated signals were selected to the display.

The ACTIVE-CAD default resolution is 10 picoseconds. To simulate with this resolution a 50 nanoseconds Short step, ACTIVE-CAD would have to simulate 5,000 cycles, which may take quite long to execute, particularly for large designs.

NOTE: To speed the simulation, set the simulator resolution to 1 ns when simulating in the **Unit Delay** mode.

Glitch Simulation

To select the **Glitch** mode, click on the **Mode** button in the simulation (#13 in Figure 2-1) toolbox until it displays **GL**. The **Glitch (GL)** simulation mode is based on unit propagation delays, which means that all components have the same delays from input to output. The value of this standard delay can be set anywhere from 10ps to 4ms. The default value is equivalent to the **Short** step setup, which is available from the **Simulation Step** option in the **Utilities** menu. The **Long** step setup has no effect on the simulation which proceeds at the **Short** step increments.

The **Glitch** mode is used for functional verification of designs that have oscillating feedbacks, and cannot be simulated in the functional mode. No timing errors are reported in the **Glitch** mode. However, any glitches will be reported and displayed in the waveform window, providing that the associated signals were selected to the display.

NOTE: To account for unit propagation delays through ICs, the timings do not represent the actual timing scale but rather the sequence of events. This may result in an asymmetrical display of clocks!

Timing Simulation

To enter the timing simulation mode, select the **TM** mode from the main toolbox (#13 in Figure 2-1). ACTIVE-CAD is factory set to simulate all cells and IC devices with a 10 picosecond accuracy. The timing simulator mode works only in a forward direction. When the timing mode is selected ACTIVE-CAD includes propagation delays for all device models and reports all timing errors, such as insufficient Setup or Hold times, that may occur during simulation. These timing violations are also reported for all off-screen signals and device pins because ACTIVE-CAD checks every pin of every device during each clock cycle. To search for errors, use the **Error Search** button (#14 in Figure 2-1) and the associated error search arrows (#15 in Figure 2-1)

The timing simulation mode is used as the last stage in design analysis. Generally, it should be run with the same test vectors that have been used for functional or glitch simulation. This will produce results that are easier to understand and will simplify tracing of design errors.

The timing mode automatically simulates device internal propagation delays and ASIC or board layouts, if it is provided within the post-layout netlists. The timing propagation delay values can be changed to MIN., MAX, AVG. and as a % of the MAX. value. This allows you to test critical paths and emulate the effect of voltage, temperature and loading factors.

To change the propagation delay values, click on the **Edit Timing Specification** option within the **Patching** menu. Then double-click on the device whose propagation delay values you want to change.

Summarizing timing simulation:

- ☐ The timing option must be installed to simulate designs in the timing mode (standard on SUSIE-CAD/PRO and ACTIVE-CAD)
- ☐ The timing resolution is recallable from 10 picoseconds to 1 millisecond (use the **Simulation Precision** option in the **Options** menu)
- ☐ All models produce timing information if the timing option is available
- ☐ Layout delays are simulated if provided via the post-layout netlist
- ☐ Propagation delays can be rescaled for each device and hierarchical macro (use the **Patching/Edit Timing Specification** option)
- ☐ Each device pin is checked during each clock cycle for timing violations and bus conflicts
- ☐ All errors are explicitly listed and saved in an error report file
- ☐ The **Error** button (#14 in Figure 2-1) and the associated search arrows (#15) allow for instant error location

Selective Simulation

Since ACTIVE-CAD is a real-time simulator, you can deselect and select any design section for simulation while the simulation is in progress. This speeds the simulation of large designs because only the selected design sections have an effect on the simulation time.

The **Selective Simulation** option is available from the **Options** menu (ACTIVE-CAD only). When selected, it automatically displays the design hierarchy in the **Scan Hierarchy** field and IC devices in the **Chip Selection** field. All selected or enabled design macros and ICs are colored dark for model available. All disabled design macros and ICs are colored white for empty symbol and are not simulated.

You can enable/disable any design macro or IC by placing the cursor in the **Scan hierarchy** window and clicking the left mouse button. ACTIVE-CAD will display, in the **Chip Selection** window (Figure 2-55), all ICs and macros at that selected hierarchical level. You can toggle each individual IC device and macro with the left mouse button and enable or disable simulation of the item.

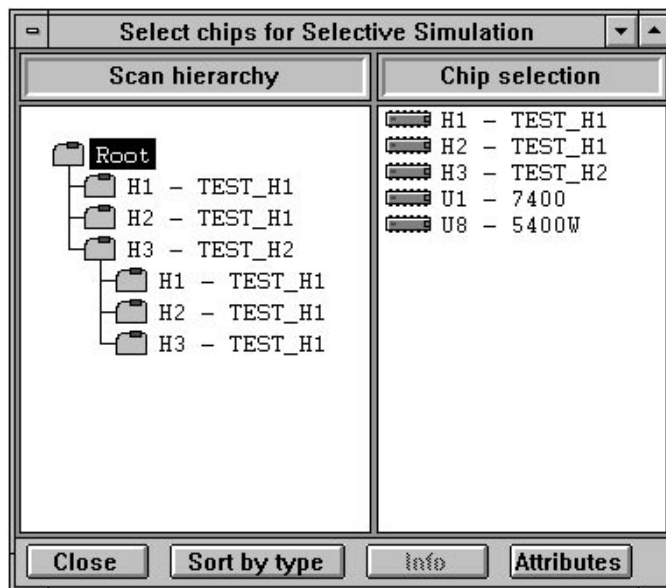


Figure 2-26. Selective Simulation.

By placing the cursor in the **Chip Selection** window and pressing the right button, you can display the selective simulation options, such as **Disable All Chips**, **Enable All Chips**, **Disable Entire Project**, etc.

The **Disable Entire Project** option allows you to quickly select only a few devices from a project that may involve thousands of devices (first disable all devices and then select the few that are needed).

Disabled devices and macros behave during simulation like empty sockets. All their pins (inputs and outputs) become inactive and change to the high impedance state. You can then manually assign test vectors to the output pins of the empty devices to emulate their behavior.

Selective simulation is patented technology designed to speed the simulation process of very large designs. If the simulation of your design is too slow for you to interactively monitor its progress, you can, with the help of the **Selective Simulation** option, enable and simulate only small sections of the design. You can start selective simulation by selecting the design front-end. Its outputs can be recorded, and used as inputs for another section. Next, you can disable the first section, which no longer has to be simulated, and use the saved test vectors as inputs to the second design section. This way you can debug the entire design section-by-section, and simulate each section at high simulation speed, independent of the overall design size.

Long Simulations

The **Simulation Stop** option in the **Options** menu allows you to run a long simulation for a specified amount of time when the **Start** button is activated. Figure 2-27 shows a window for setting the simulation time and enabling the **Stop** button (#16 in Figure 2-1).



Figure 2-27. Simulation Stop Setting.

When the **Sim. Till End** option is selected, simulation will run until the end of the specified **Simulation Running Time**. Always select the **Stop** button active so that you can stop the simulation when required. The only time you may wish to disable the **Stop** button is when you are leaving the office and do not want anybody to interfere with your simulation.

Analyzing Simulation Results

Table 2-2 lists all available signal logical states, their descriptions and symbols displayed on a waveform diagram.

Analyzing waveform displays

ACTIVE-CAD has many features that ease waveform diagram analysis:

- ☐ You can zoom in and zoom out on a selected waveform section using the buttons located at the top left corner of the waveform window (#15 in Figure 2-2). These buttons set the waveform scale or ruler to a predefined setting.



There is a dynamic zoom feature which is active when the ruler or waveform scale is displayed at the top of the waveform window. Position the



cursor over the ruler, click at the beginning of the area of interest and drag the blue line to the end of the desired waveform area. When you release the mouse button, the selected area is enlarged to a full size window. To scale back the signal waveforms, click on the **Scale Adjusting** field (#15 in Figure 2-2)

Table 2-2. ACTIVE-CAD Logical States

State Name	Symbol	Description
Low		Strong Logical Low state, e.g.. output of a TTL gate.
High		Strong Logical High state, e.g.. output of a TTL gate.
High Impedance		Tri-Stated output or unconnected input.
Unknown		Strong Undefined High or Low state, e.g.. initial state of a TTL flip-flop.
Resistive Low		Weak Logical Low state, e.g.. Open Emitter output in Low state.
Resistive High		Weak Logical High State, e.g.. Open Collector output in High state.
Resistive Unknown		Undefined Resistive Low or Resistive High state.
Output Conflict		Indicates that there is a bus conflict in the node (High and Low at the same time)
High Voltage		This logical state indicates that there is a high voltage at the pin. E.g. +12V, -5V, etc. etc.
Reference Voltage		The Reference voltage used in ECL technology
Unknown Activity Low		Low or Resistive Low or High Impedance. This state is generated by tri-state ICs when the tri-state control pin is undefined.
Unknown Activity High		High, Resistive High or High Impedance. Similar to the above.
SV High		Power Voltage High (e.g.. VCC)
SV Low		Power Voltage Low (e.g.. GND)
SV X		Power Voltage Unknown

To quickly find red marked timing errors, select the **Error** option in the **Search** button in the main toolbox (#14 in Figure 2-1). Each activation of

the  or  **Search** buttons will move the cursor to the nearest timing error location.

- ☐ You can define and instantly locate selected signal combinations called TAGs (see the **TAG Conditions** section of this manual). The location of these TAGs can be quickly searched by using the **TAG** option in the **Search** button, located in the main toolbox. Each activation of the  or  **Search** buttons moves the timing cursor to the next TAG position.
- ☐ To scroll a waveform diagram, use the scroll bars located at the bottom of each waveform window. You can click on the scroll bar arrows to scroll one division at a time in the desired direction. Use the **Scale Adjusting** field (#15 in Figure -2) to set the division in nanoseconds. You can also click on the scroll bar body to scroll the waveforms one page (one window size) in the desired direction. To quickly move to any screen area, you can drag the scroll bar button to the desired waveform location, e.g., by dragging the bottom scroll bar button all the way to the left you can instantly display the beginning of the waveform diagram.

Symbols and Colors

The ACTIVE-CAD signal color code is as follows:

- ☐ input signals are displayed in *black*
- ☐ all active outputs are in *blue*.
- ☐ signals in logical level conflicts, e.g., High connected to Low, are in *red*
- ☐ signals overriding others during editing are marked in *green*.

Measuring time intervals

ACTIVE-CAD allows you to precisely measure any timing relationship between displayed signal waveforms. To enter the waveform measurement mode, select the **Measurements/Measure On** option from the **Waveform** menu. Notice that the cursors shape has changed to indicate the waveform measurements mode.

To measure the delay between two events, position the cursor on the first event and click the mouse button. Place the cursor over the second event and click the mouse button again. Green reference marks will appear to denote the events involved in the measurements. ACTIVE-CAD will force an empty signal line to make a room for the measurement data, which will be displayed in red. If the displacement between events is too small, the measurement will not be displayed and you will need to res-

cale the display using the **Scale Adjusting** field (#15 in Figure 2-2) till the measurement data is displayed.

The cursor in the measurements mode has a *Snap to the Edge* feature, which allows you to measure time delays with better precision than that from using the waveform scale setup. For example, if the scale is 10 ns and the measured transitions are at 1023.5 ns and 1027.3 ns, then because the cursor snaps to both transitions, a correct delay of 3.8ns between both signals will be measured and displayed, despite the 10 ns display scale.

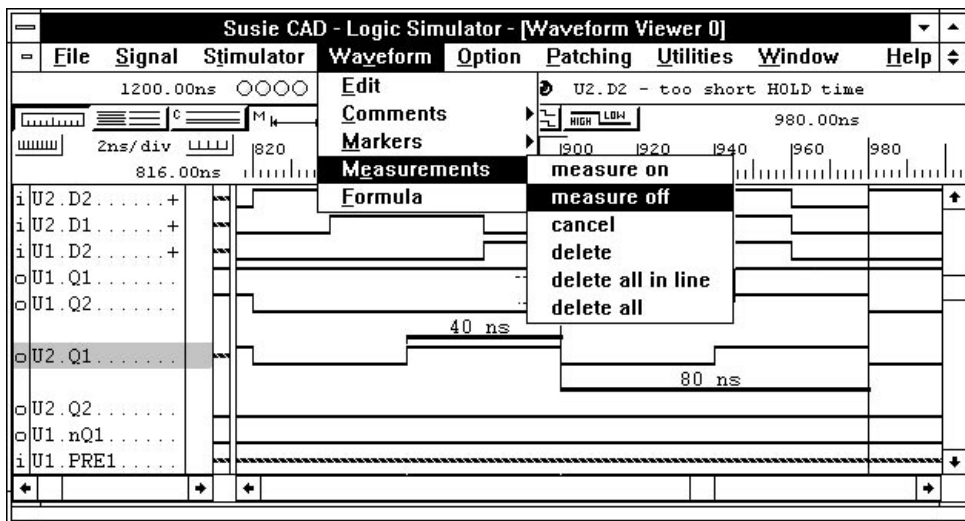


Figure 2-28. Measuring time intervals.

To exit the measurement mode, invoke the waveform menu again, and select the **Measurements/Measure Off** option (Figure 2-28).

To toggle the measurement display ON/OFF, click on the **Measurements** button  in the **Waveform Viewer** toolbar.

TAG conditions

Analyzing simulation results can be time consuming, especially with long simulations. Finding the desired signal combination or sequence by manually scrolling the waveform diagram can be tedious and cumbersome, particularly since you may need to watch for more signals than can be displayed directly in the waveform viewer. To help with this problem, ACTIVE-CAD comes with the **TAG** feature, which allows you to define any set of signal conditions and then automatically search for these conditions (TAGs) throughout the entire display.

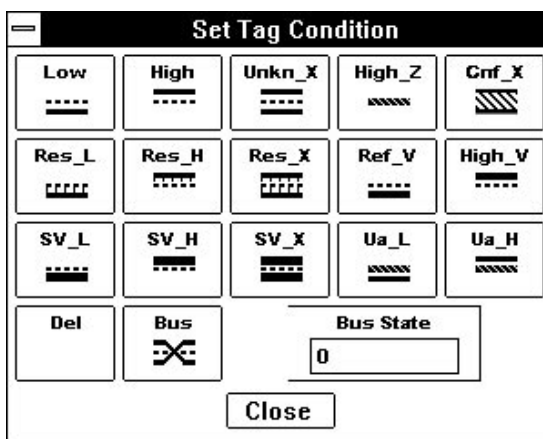


Figure 2-29. Set TAG Condition Window.

First, display the **TAG** column in the Waveform Viewer by selecting the **TAG** option in the **Utilities/View** menu. Next, select the **Set TAG Condition** option in the **Options** menu, which displays the **Set TAG Condition** window (Figure 2-29). Following this, select (highlight) a signal in the **Waveform Viewer** that you want to assign a TAG condition to, and click on the desired logical state in the **Set TAG Condition** window. Repeat the same procedure for all the signals that you want to include in the TAG.

If you define multiple signals in the TAG condition, ACTIVE-CAD will be looking for the AND combination of these conditions. For example, if you select the Low state for one signal and Unknown for another, ACTIVE-CAD will search for all situations in which the first signal is High and the other one is in the Unknown state at the same time. An example of the **Waveform Viewer** is shown in Figure 2-30.

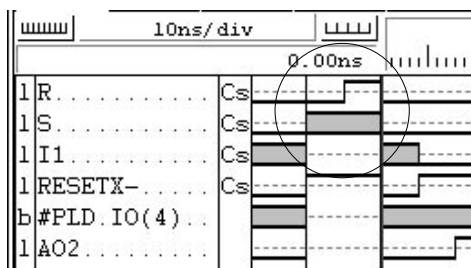




Figure 2-30. Example of the TAG settings.

You can also include signal transitions in the TAGs. Instead of looking for a certain signal logical state, you can instead look for its transition

from one state to another. To define a transition in the TAG condition, first click on the button in Figure 2-29 that represents the logical state before the transition and then on the button representing the state after the transition. This will display the transition next to the signal name in the **Waveform Viewer**. For example, if you first click on the High state and then on Low, you will define a transition from High to Low. You can define TAG conditions that include both stable signal states and transitions. If you have defined a transition for a selected signal and want to change it to a stable signal, click on the same logical state twice.

Searching for a TAG

You can use the main simulation toolbox to search for the TAGs defined in the **Waveform Viewer**. Using the **Search Mode** button (#14 in Figure 2-1), select the **TAG** search mode (This button combines several modes and you may have to click on this button several times before the **TAG** mode is displayed). Next, click on the location from where you want to start your search. The blue vertical cursor will move to that location. Then use the **Search Right**  and **Search Left**  buttons in the main simulation toolbox to find TAG condition in the **Waveform Viewer**. Each click of the Search Left or search Right buttons will move the blue vertical cursor to the next Tag condition.

If the Tag includes only signal levels and does not include any events (signal transitions) then the search operation will stop at the first occurrence of the Tag signal levels in the searched direction. Such Tag is called dual-Tag because it produces two blue vertical cursors for the same signal conditions, one for searching from the left direction and another one for searching from the right direction.

Inserting Comments

You can insert comments between waveforms. These comments may prove invaluable when you try to analyze the design at some later date.

To enter a comment, click the cursor at the desired location within the waveform diagram and select the **Comments/Add** option from the **Waveform** menu. In the **Add New Comment** window (Figure 2-31) enter the text to be displayed at the selected location and press the **Add** button. The text will automatically be placed at the selected screen location.

If you want to edit an existing comment, follow the same procedure. First, move the cursor over an existing comment and press the left mouse button. Then invoke the **Comment/Edit** option from the **Waveform**

menu. When the current comment is displayed in the comment editor window, edit it and click on the **Add** button.

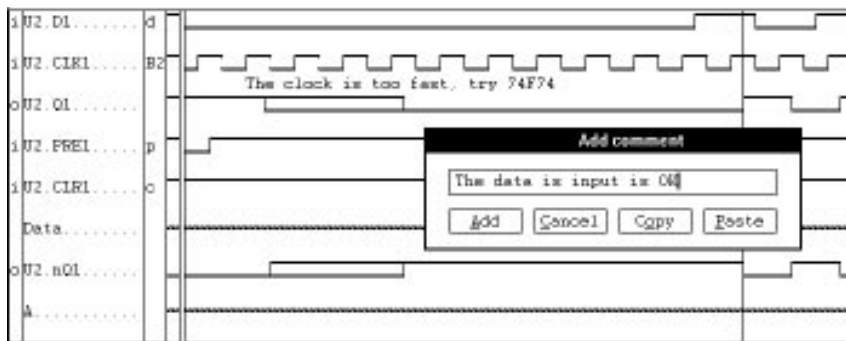


Figure 2-31. Placing comments in the timing diagram.

To delete a comment, click on it in the **Waveform Viewer** and then select the **Comment** option from the **Waveform** menu. To delete all comments in the Waveform Viewer, click on the **Delete All** option. To delete all comments in the selected line, click on the **Delete All in Line** option.

Memory Editor

The **Memory Editor** option allows you to view and edit the contents of memory devices. You can monitor the data as it is written into a memory location during simulation and modify it when needed.

Select the **Memory Editor** option, located in the **Utilities** menu. The **Select Block/Chip for Memory Editor** window will appear and list all memory devices at the current hierarchical level. Select the desired memory device. The **Memory Editor** window will display its contents and addresses in the formats you choose from the **Display Options** window, which is invoked by clicking on the **Display** button in the Memory Editor window (Figure 2-32).

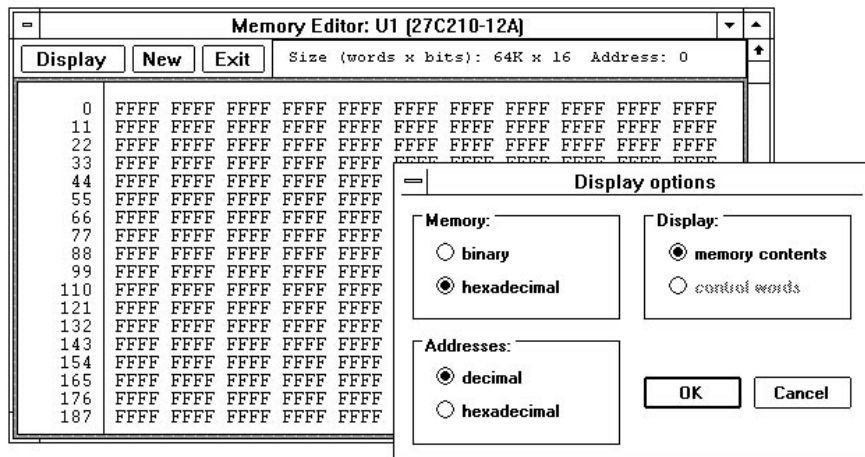


Figure 2-32. Memory Editor window.

Save/Load Simulation Results

Simulation Files

Simulation files represent work in progress. They contain the design status that existed in the ACTIVE-CAD simulator after the last simulation step. This simulation status is stored in a set of design tables that contain information about device connections, waveform diagrams, states of sequential logical devices, JEDEC fuse maps, hex files, position of switches and jumpers, simulation settings, etc.

Simulation files represent a concrete schematic design implementation. If that schematic undergoes any changes, a new design file has to be generated for the schematic.

Simulation data is created on-the-fly during loading of a schematic netlist. This design data is continuously updated with new test vectors, hex data, JEDEC fuse maps, etc., as they are being loaded into the simulator. You can save this design information at any time using the **Save Simulation** option.

You can load and save design files by selecting the **Load Simulation** and **Save Simulation** options, located in the **File** menu. Upon selection of the load or save design options, ACTIVE-CAD will display a list of design files that have already been saved within the current project (Figure 2-33). If you want to save a design, move the cursor to the **Simulation Name** box and enter the desired design name. To load a design, select a

design file name from the list displayed in the design window (Figure 2-33). ACTIVE-CAD will always look for design files with the *.DES file name extension, stored in the directory specified in the ACTIVE-CAD configuration. These files will be displayed only if they belong to the current project.

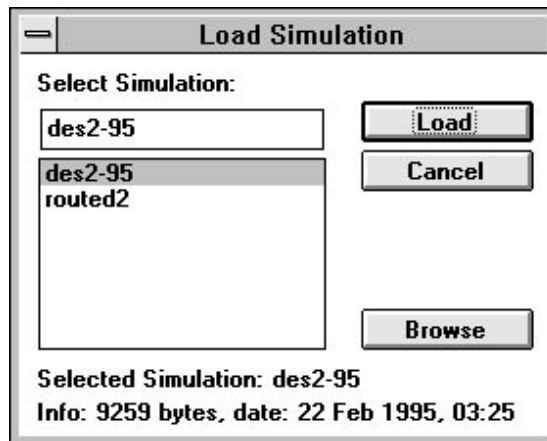


Figure 2-33. Load Simulation Window.

NOTE: Loading a simulation file from another project is not recommended because it can be based on a different schematic or netlist. However, if you wish to do so, you can use the **Project Resources** option in the **Project Manager** menu and add any desired simulation file to the project.

Test Vector Files

Test vector files are stored in a binary format. They include all signal names listed in the waveform windows and the associated test vectors (waveforms). ACTIVE-CAD also saves the location of the waveform cursor, waveform scale, position of the waveform on the screen, etc. The information contained in a test vector file is complete and sufficient to restore and display waveforms that have been generated by a previous simulation session.

If no design or netlist has been loaded, then the test vector file can only be used for viewing the previously generated simulation results.

If a design or netlist file has been loaded prior to loading a test vector file, then the test vector file is automatically associated with the design

signals, and can be used not only for viewing the previous simulation but also for new simulations of the design.

Loading and saving test vector files is performed from the **Load Test Vectors** and **Save Test Vectors** options in the **File** menu.

ASCII Test Vector Files

To allow easy data exchange with other CAE tools, ACTIVE-CAD both generates and accepts ASCII files. The ACTIVE-CAD ASCII format is similar to other ASCII formats that are popular in the industry. The specification of the ACTIVE-CAD ASCII format is provided in Appendix B.

To load and save ASCII test vector files use the **Load ASCII Test Vectors** and **Save ASCII Test Vectors** options in the **File** menu.

Error Reports

Timing Violations

ACTIVE-CAD automatically checks every device/cell pin during each clock cycle for timing violations. All violations are registered, together with the type of violation, time of occurrence and the size of the timing violation. For example, ACTIVE-CAD can detect a Too Short Setup Time error and show how much time was missing for proper device operation. This data is subsequently available for various design error reports.

NOTE: Timing Errors are reported by VHDL models, each of which can detect a number of errors specific to the device.

The **Error Reporting** option, in the **Options** menu, is used to save and display timing violations. All detected errors are automatically saved, and can be displayed by double-clicking on the the icon called **ACTIVE-CAD Messages**.

Bus Conflicts

ACTIVE-CAD automatically checks every device/cell pin during each clock cycle for bus conflicts. All such violations are recorded, together with the time of occurrence and conflicting pins and voltage levels. This data is subsequently available for various design error reports.

The **Error Reporting** option, in the **Options** menu, is used to save and display bus conflicts. To display a list of ACTIVE-CAD messages, double-click on the **ACTIVE-CAD Messages** icon. For more information on errors, see *Design Error Handling* in Chapter 1.

Creating Reports

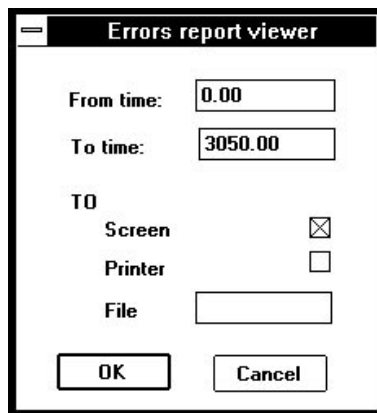


Figure 2-34. Dialog box for creating error reports.

ACTIVE-CAD provides extensive design error reporting that speeds design analysis and error location. To create a post-simulation error report, select the **Error Viewer** option from the **Utilities** menu. Following this, select the time range for error reporting from the dialog box shown in Figure 2-34, and click on the OK button.

This will invoke a **Report** window with a chronological list of errors as shown in Figure 2-35. Using the options in Figure 2-34, this report can also be saved to a file or printed.



Figure 2-35. Error Report window.

Design Patching

Editing Propagation Delays

ACTIVE-CAD libraries contain timing specifications for all their IC parts. Maximum, minimum and average values of specific timing parameters are accessible through the **Library Manager** and are based on manufacturers data books. These parameters can be edited at any time during simulation. However, only in the timing mode are these values used by the simulator.

NOTE: Editing propagation delays has no effect in SUSIE-CAD/HIER, because it only has functional and glitch simulation modes. However, if equipped with the TIM option, it will respond to the propagation delays.

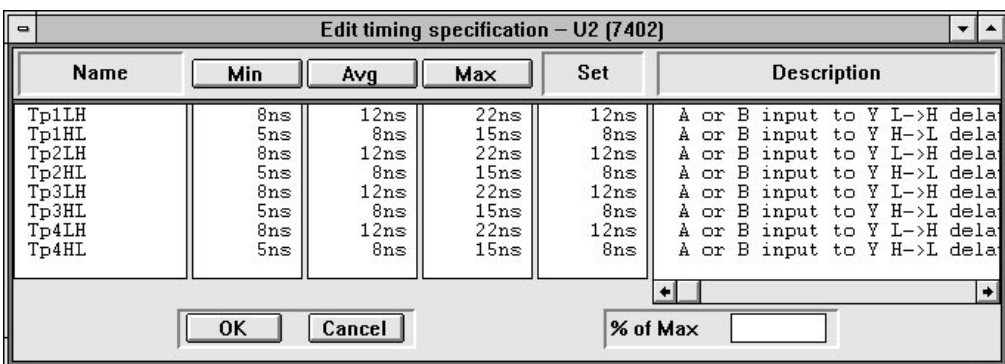


Figure 2-36. Editing propagation delays.

Initially, all ICs are loaded with their average (typical) values. However, using the propagation editor you can change these values. To access the propagation editor, select the **Edit Timing Specification** option in the **Patching** menu. A listing of all parts and macros at the current hierarchy level will be displayed. By selecting a desired hierarchical level and IC, you can display its timing specification table (Figure 2-36). You can edit this table to suit your design requirements. For example, you can set the propagations to Min, Max, Avg. or any % of the Max. In addition, you can edit each parameter individually in the **Set** column.

You can create your own parts by editing in the **Library Manager** the **Min.**, **Avg.** and **Max.** timing parameters of devices supplied by ALDEC.

Please refer to the Library Manager (Chapter 8 in *Schematic Editor Users Guide*) for details on modifying simulation libraries.

Editing Device Timing Parameters

In addition to changing individual parameters, the ACTIVE-CAD simulator allows you to change propagation delays of devices, selected design sections, hierarchical design levels and even entire designs to:

- * Maximum values
- * Minimum values
- * Average (typical) values
- * N % of maximum (even below minimum or above maximum values).

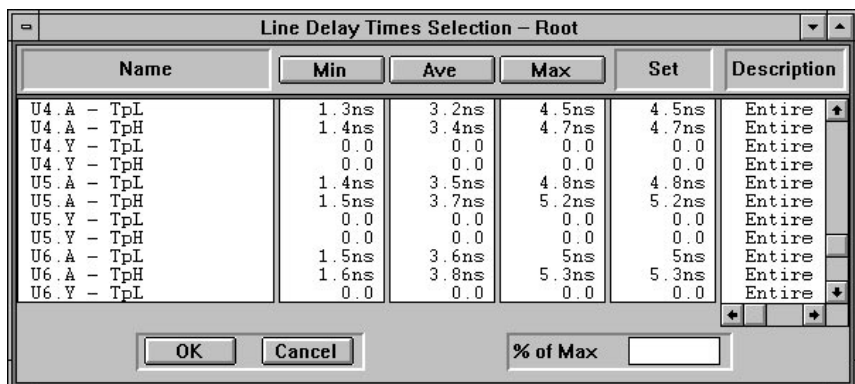
To change the propagation delay parameters of any device, select the following items in sequence:

- ☐ **Patching** menu
- ☐ **Edit Timing Specification** option
- ☐ Double-click on the selected hierarchical level to display its parts
- ☐ Double-click on the selected chip. When a menu appears, select either **Min**, **Avg.**, **Max** or enter the % of **Max** into the **Edit Timing Specification** window.
- ☐ You can edit timing parameters in the **Set** column of Figure 2-36.

The above options allow you to set and test your design with the worst-case timing conditions. You can set propagation delays to any values and instantly resimulate the entire design to verify the effect of these changes.

Editing Line Delays

Line delays are the delays between pins of the netlist components and represent routing delays in ASICs, FPGAs and PCB layouts. ACTIVE-CAD simulates line delays and IC propagation delays separately. If a design netlist includes line delays, you can view and edit these delays using the **Change Line Delays** option in the **Patching** menu.



Name	Min	Ave	Max	Set	Description
U4 .A - TpL	1.3ns	3.2ns	4.5ns	4.5ns	Entire
U4 .A - TpH	1.4ns	3.4ns	4.7ns	4.7ns	Entire
U4 .Y - TpL	0.0	0.0	0.0	0.0	Entire
U4 .Y - TpH	0.0	0.0	0.0	0.0	Entire
U5 .A - TpL	1.4ns	3.5ns	4.8ns	4.8ns	Entire
U5 .A - TpH	1.5ns	3.7ns	5.2ns	5.2ns	Entire
U5 .Y - TpL	0.0	0.0	0.0	0.0	Entire
U5 .Y - TpH	0.0	0.0	0.0	0.0	Entire
U6 .A - TpL	1.5ns	3.6ns	5ns	5ns	Entire
U6 .A - TpH	1.6ns	3.8ns	5.3ns	5.3ns	Entire
U6 .Y - TpL	0.0	0.0	0.0	0.0	Entire

OK Cancel % of Max

Figure 2-37. Line delays setting.

The **Scan Hierarchy** window that appears on the screen allows you to select any desired hierarchy level. Double click on the desired hierarchical level and the **Line Delay Table** (Figure 2-67) will show all line delays at the selected hierarchical level. You can edit any delay value in the **Set** column by double clicking on it, entering the desired value, and then pressing the **Enter** key.

By changing line delays, you can emulate different layouts and their effect on simulation results.

To view the effect of line delays, select to the display screen a device output pin that drives the selected node and an input pin that receives this signal. After some simulations you will observe that timing transitions on the input pin are delayed in reference to its source (output pin).

NOTE: If you select to the screen the signal names of the nodes that have line delays, the displayed waveforms will represent the timing of the output pins that drive the selected nodes.

Change Technology

ACTIVE-CAD allows you to instantly change an IC technology. To replace a part with another one, select the **Change Technology** option in the **Patching** menu. When a **Change Technology** window appears, select an appropriate design hierarchy in the **Scan Hierarchy** window. Then double click on the selected IC in the **Chip Selection** window. When a table of suggested IC replacements appears, select the desired IC device name, and press the **OK** button.

The only restriction is that the replaced part has to be of the same type (logic behavior) and has to be in the same library as the original part (TTL, PLD, CMOS, etc.). For example, you may replace a 74LS74 with a 74F74 but not with a 74LS00 or 22V10. If the new part exists in the same library as the original part, a new timing specification will be loaded to the simulator and displayed on the screen.

Changing Generic Values

All IC models are set to their typical timing parameters by default. In addition to the **Edit Timing Specification** option that allows you to change selected timing values for simulation, you can also set global design conditions by using the **Change Generic Values** option in the **Patching** menu. This option allows you to select any IC device and set its operating temperature in the window shown in Figure 2-38. If you change the temperature setting, ACTIVE-CAD will automatically calculate a new timing parameter for the chip. This option works only with selected IC models that have a built-in temperature specification.

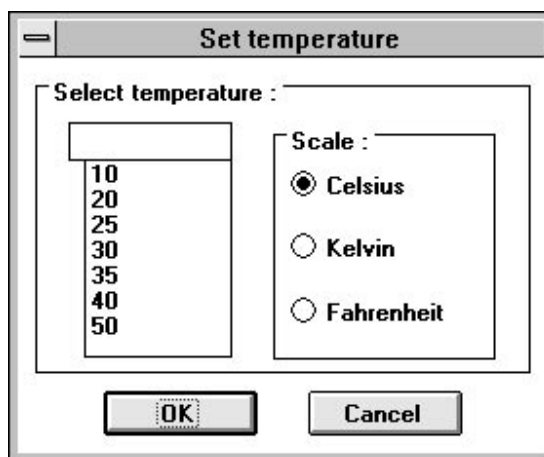


Figure 2-38. Changing Generic Values of IC devices.

Toggling Switches

If you use switches or jumpers in your design, ACTIVE-CAD allows you to change the position (toggle) of these elements, so that you can test your design with different settings. When you select the **Switch Settings** option in the **Patching** menu, the **Switch Settings** window will list all switches and jumpers at the selected design hierarchical level. Select the

desired switch and press the **OK** button. This will display a window with the graphical representation of the selected switch.

Using the **Toggle** button, you can set the desired switch position and toggle the jumpers. Double clicking on the desired switch position forces its setting.

If you have multiple switches in the design, and need to toggle them frequently, you can open and then minimize the switch windows for all switches that you plan to use. When you click on the minimized switch window, it will display the **Toggle** option, which when selected, toggles the switch and displays the new switch setting. If you use switches in a hierarchical design, you can also use the **Hierarchy Scan** option to verify on which level the switch is located. The list of supported switches and jumpers can be found in the Appendix C.

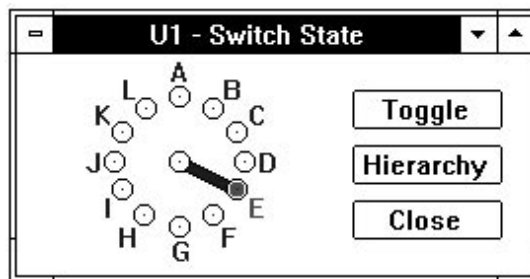


Figure 2-39. Setting Rotary Switch Position.

Passive Components

ACTIVE-CAD does not directly simulate analog circuits. You can, however, create a Spice netlist and simulate analog circuits with Intusofts Is-SPICE software product.

The only analog components that are recognized by the digital simulator are resistors, resistor packs and inductors. Each inductor is simulated as a short connection of two nets. A resistor may be simulated as a Pull-Up, Pull-Down or short connection, depending on how it is connected. The Pull-Down function will be used when one of the resistor pins is connected to the GND (or other ground signal). The Pull-Up function is used when one of the resistor pins is connected to VCC or any other power signal. The Pull-Up and Pull-Down functions will also work with the OE (Open-Emitter) and OC (Open-Collector) outputs. The Pull-Up and Pull-Down resistors can be used on both input and output pins. The list of

passive components recognized by ACTIVE-CAD can be found in Appendix C.

NOTE: The wired OE and OC outputs will be simulated properly as wired OR and wired AND gates, respectively. The logical states generated with the Pull-Up or Pull-Down resistors are called Resistive, which means that several outputs can be connected together without causing any conflicts.

Milestones

A milestones is used to store the complete design status that exists at a selected simulation cycle. Thus, it can be used for later resimulation of the design with new design or test vector changes. A milestones saves the timing at the current simulation cycle and all internal states (registers, memories, flags, etc.) of all IC models.

The milestone data is saved on a disk as a temporary file and is always deleted when you load a new design or exit the ACTIVE-CAD simulator. When you load a milestone, the simulator restores the design to its state at the time of the milestone, preserved by the milestone save operation.

Using milestones is very convenient when you want to resimulate the design from a certain cycle with modified test vectors, new timing delays or other design changes.

You can save as a milestone every significant simulation point and use it later on for instant resimulation. Also, you can automatically save milestones during simulation, and if a design problem is detected, you can always restart the simulation from a point just before the problem occurred.

Milestones allow extensive design analysis. You can resimulate a modified design from a selected milestone and request ACTIVE-CAD to automatically display the effect of the latest design or test vector changes on the selected signal(s).

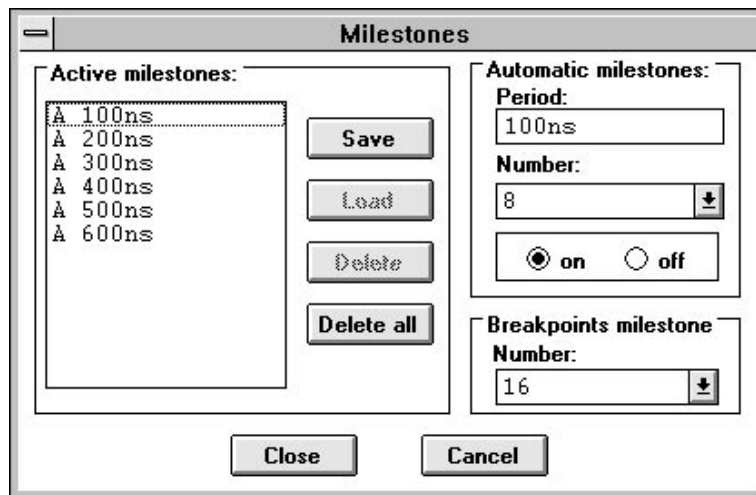


Figure 2-40. Saving Milestones.

Saving Milestones

Manual milestones

You can manually save any selected design condition as a milestone by pressing the **Save** button shown in Figure 2-40. The saved milestone will appear on the Waveform Viewer as a red dot on the ruler (time scale display at the top of the screen).

Automatic milestones

In addition, you can select the **Automatic Milestones** options, in which you can save milestones at set time intervals. You specify both the number of milestones in the set (up to 32) and the time interval between each. Since each milestone uses hard disk space, keep the number of milestones in the set to a minimum. Once the set is full, any new milestone will overwrite existing ones, starting with the earliest milestone in the set.

To turn on automatic milestones, use the **On** option located in the **Automatic Milestones** field. Similarly, select the **Off** option to stop saving the automatic milestones.

Breakpoint-driven milestones

ACTIVE-CAD can automatically save a milestone upon detecting the desired signal conditions. To activate the breakpoint-driven milestones, follow this procedure:

- ☐ Select the **Breakpoint** option from the **Utilities** menu.
- ☐ Select the **Edit** option from the **Breakpoint** menu; The **Breakpoint Conditions** window appears.
- ☐ Click on a Condition field 0-F (vertical columns); The selected Condition turns red.
- ☐ Click on a signal in the **Signals** field
- ☐ Click on the **States** button in the **Breakpoint Conditions** window
- ☐ Assign a logical level to the signal by clicking on the corresponding button in the **Breakpoint States** window; To assign a signal transition, assign two logical states to the same signal. The signal transition is displayed in the **Breakpoint Conditions** window.
- ☐ After the signal Condition have been set, click on the **Edit** button
- ☐ A **Breakpoint Edit** window will appear, listing all conditions; Double-click after the **Then** command.
- ☐ A new window listing breakpoint editor instructions appears. Click in that window on the **Mark Breakpoint** instruction, enter the breakpoint name into the Argument field and click on the Set button.
- ☐ Select additional breakpoint instructions if needed and click on the **Close** button to exit the window.
- ☐ Click on the **OK** button in the **Breakpoint Edit** and then in the **Breakpoint Conditions** windows. ACTIVE-CAD automatically sets the breakpoint to the On condition and will save a milestone each time a breakpoint is found.

Since the milestones could fill up the hard disk very quickly, select the **Milestones** option from the **Options** menu and enter in the **Breakpoint Milestone** field the maximum number of milestones to be saved.

Loading Milestones

You can display all available milestones by selecting the **Milestones** option from the **Option** menu. The **Active Milestones** field in Figure 2-40 displays a time at which the design was saved. To load a milestone, invoke the **Milestone** option, select a milestone from the **Active Milestone** table and then select the **Load** option. You can delete selected milestones using the **Delete** selected milestone or **Delete All** milestones option.

Printing Simulation Results

Page Setup

To print a timing diagram, you need to specify its format with the help of the **Page Setup** option in the **File** menu. The window shown in Figure 2-41 allows you to select the following options:

The **Waveform Spacing** section allows you to print extra lines of information between the waveforms. These lines may contain your comments and and/or timing measurements.

- ☐ **High Density Mode** - disables all comments and measurements. It does not delete any empty lines
- ☐ **Delete Empty Rows** - deletes all rows that do not contain any information

If neither **High Density Mode** nor **Delete Empty Rows** is selected, then ACTIVE-CAD will print waveforms in the low density mode, and will include all comments, measurements and empty lines.

The screenshot shows the 'Page Setup' dialog box with the following settings:

- Waveform spacing:**
 - ☐ high density mode
 - ☐ delete empty rows
- Options:**
 - pin/signal names:**
 - ☐ none
 - ☒ first page
 - ☐ every n-page (with a text input field below it)
 - ☒ errors
 - ☐ time measurements
 - ☐ comments
 - ☐ design changes
 - ☐ markers
- Print options:**
 - ☐ current page
 - ☒ all
 - ☐ from [] to [] ns []

Buttons: OK, Cancel, Default

Figure 2-41. Page Setup window.

The **Options** section allows you to select some additional information to be printed with the waveforms.

- ☐ The **Errors** option marks the timing errors on the printed waveforms
- ☐ The **Time measurements** option prints the waveform measurements

- ☐ The **Comments** setting allows you to print all comments inserted into a waveforms diagram.
- ☐ The **Design Change** will mark on the waveform diagram all instances of design change
- ☐ The **Markers** option will print the current location of the markers

The **Pin/Signal Names** option allows you to print the signal names with the waveform data as follows:

- ☐ **Every n-Page** setting - prints signal names on every *n*th page. You need to enter the value of *n*. For example, if you specify 4, the signal names will be printed on every fourth page.
- ☐ **None** - does not print any signal names
- ☐ **First** - prints signal names only on the first page

Print options - allow you to print selected parts of the waveforms as follows:

- ☐ **All** option prints the entire waveform diagram
- ☐ **Current page** - prints only one page of the waveform diagram starting from the beginning of the current waveform viewer;
- ☐ **From-to** - allows you to print any part of the waveform diagram by entering the beginning and end of the timing waveforms, expressed in nanoseconds, microseconds or milliseconds.

Print Setup

The **Print Setup** option, located in the **File** menu, is the standard Windows option that allows you to select a destination printer and set the device specific options.

Print

The **Print** option prints the waveform diagram on the selected printer. The settings for the waveform diagram layout are selected using the **Page Setup** option.

Saving Windows Layout

If you arrange the simulator main window, the simulation toolbox and Waveform Viewers, and you want them to be arranged the same way in the next simulation session, you can save that layout using the **Save Settings Now** option in the **Options** menu.

The **Save Settings On Exit** option saves the last screen settings before exiting ACTIVE-CAD. The last simulator windows arrangement will be restored when you restart the simulator. The layout configuration is saved in the SUSIE.INI file and can be cleared using the **Default Settings** option in the **Options** menu.

Chapter 4

MOBIC Model Builder

Introduction

The MOBIC Model Builder Compiler is based on Boolean logic equations. The logic equations can be written using any text editor. MOBIC can be used for generating functional models of SSI and MSI IC parts. It generates very fast models that can be used to simulate the components that have no models in the ACTIVE-CAD library. For modeling higher complexity ICs and for generating timing models, use the VDS (VHDL Development System).

The logic equations written under MOBIC are based on the equal sign (=) and on rising and falling clock edge equations. There are no set, reset or trigger commands under MOBIC. Flip-flops, counters, etc. are modeled by using the ALDEC clock equation format which is explained in this chapter.

Model Structure

Each IC model is composed of three (3) sections:

1. Chip declaration
2. Pins and signals definition
3. Model body

The MOBIC source code file can include more than one IC model.

Chip Declaration

The chip declaration is comprised of **CIRCUITS** and **CLASS** statements. The chip declaration must start with a **CIRCUITS** statement, followed by the IC model name or a list of functionally equivalent IC models, e.g.:

```
circuits 74LS00
```

```
circuits 74LS00, 54LS00, 54F00
```

You can write **circuits** or **CIRCUITS** but any misspelling will reject the entire model. For example, **CIRCUIT** will reject an IC model.

The **CLASS** statement defines the type of logic, e.g. TTL, ECL, etc. It will help later in the ACTIVE-CAD simulator to detect any illegal connectivity between various logic families. The class statement cannot be omitted. An example of a class statement is:

```
class TTL
```

Pins and Signals Definition

The pins and signals definition is comprised of **PINS** and **SIGNALS** statements. The **PINS** statement lists all I/O pins and defines their I/O types, e.g. IN (input), OUT (totem-pole output), TRI (tri-state output), etc. A list of pin types is provided in the *MOBIC 6.0 Specification* section.

A signal is a logic variable to which you can assign logic values to store them in the model. The **SIGNALS** statement lists all signal names that are used internally by the IC model. Signals are very useful for storing temporary values, like expression results or internal logic states. Even if no internal signal names are used, the **SIGNALS** statement must be included, e.g.:

```
SIGNALS;      (empty space)
```

You do not have to use the **SIGNALS** statement when you are creating a new IC model based on an existing one. Instead, enter the **USE** statement in place of the **SIGNALS** statement.

Model Body

The model body starts with the **EQU** statement and ends with **END;** statement, which ends the IC model. The body of the IC model is com-

prised of Boolean logic equations. Following is an example of a model body:

```
EQU
  1Y = -(1A*1B);
END;
```

The specification of syntax, logic states, operators, etc. is included in the *MOBIC 6.0 Specification*, later in this Chapter.

Combinatorial Model Conventions

The combinatorial IC parts, such as gates, decoders and similar devices that do not have storage elements such as flip-flops, are modeled directly with the help of the equations. To create a combinatorial IC model define each IC output pin as a logical function of input pin states. For example:

AND Gate
$1Y = 1A * 1B$

OR Gate
$1Y = 1A + 1B$

Sequential Model Conventions

Sequential circuits are driven by rising and falling clock edges. A rising edge means that a signal is changing its logical state from LOW to HIGH. A falling edge is a change from HIGH to LOW. All actions in sequential circuits are related to rising, falling or both edges. It is important then to detect these edges in the model and be able to write logic expressions based on the signal changes. The following is an example of a sequential model (D Flip-Flop):

```
CIRCUITS 7474;
CLASS TTL;
PINS  CK:in:3,  \ clock input
        D:in:2,   \ data input
        Q:out:5;   \ data output
SIGNALS edge,   \ variable used to detect edges
        oldclk;   \ variable used to store previous CK input state
```



```
EQU  
edge = CK * oldclk;    \ rising edge detection  
IF edge EQ HIGH      \ if rising edge occurred pass the D input to the  
Q output  
    Q = D  
ENDIF  
oldclk = CK;    \ you have to store the current CK state for the next  
simulation cycle  
END;
```

In each simulation cycle, the current state of the clock input pin **CK** is compared to its logical value in the previous cycle, stored in the **oldclk** variable. The comparison is performed using the **AND** operator (*). The result of this comparison is assigned to the edge variable. Edge is **HIGH** only when **oldclk** is **LOW** and **CK** is **HIGH**, which is equivalent to the rising edge on the **CK** input.

If the rising edge occurred, the **IF** statement copies the logic state on the **D** input to the **Q** output pin. Afterwards, you have to save the **CK** input state in the **oldclk** variable for the next simulation cycle.

If you want to detect a falling edge instead of the rising one, use the following expression:

```
edge = oldclk * CK
```

If you want to detect both falling and rising edges use:

```
edge = oldclk # CK    \ exclusive OR of both signals
```

Remember that ACTIVE-CAD is using a 12-value logic and you should consider all possible combinations of the input pins. For example you may want to test input pins for **Unkn_X** or **Hi_Z** state.

Compiling a model

The MOBIC compiler is a program that reads the MOBIC Language source file, compiles models, generates error prompts puts the models into the current project library once the models are compiled successfully.

To run the MOBIC compiler, double-click on the MOBIC icon within the ACTIVE-CAD program group.

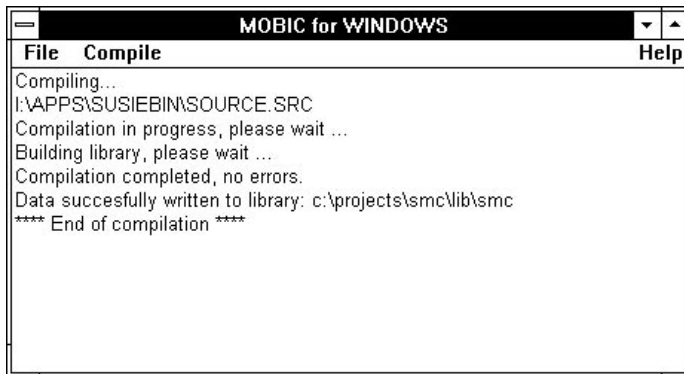


Figure 4-1. MOBIC Compiler window.

The source file can be edited in any text editor, e.g. Windows Notepad. You can save the source file in any directory and have any filename extension, e.g. SOURCE.SRC. To open the source file for compilation, select the **Open** option within the **File** menu and select the source file.

Next, select the **Options** from the **Compile** menu and choose whether you want to put the compiled models into the library for simulation or whether you want to generate the source code debugger information, and whether you want to generate symbolic code.

The simulator's source code debugger allows for detailed monitoring of the models operation.

To start the compilation of your models, use the **Start Compilation** from the **Compile** menu. The compiler displays the progress of the compilation in the MOBIC main window. If for any reason, you need to stop the compilation, use the **Stop Compilation** option from the **Compile** menu.

After you compile and insert your models into the project library, you can use them in both the simulator and the schematic editor. The schematic symbol is generated automatically and can be edited using the symbol editor.

MOBIC 6.0 Specification

Set of characters

Allowed characters in MOBIC are:

letters: **a-z** and **A-Z**

digits: **0-9**

special char.: **_ / @ \$ % ! . ^ ~ ? [] { } " _**

operators: **+ - * # () =**

separators: **; :**

blank space

comment markers: *** **

Signal and Pin Names

Names can include any letter, digit or special character in any order. If a pin/signal name includes any of the operator symbols or is to be case sensitive, the name must be enclosed in quotation marks (" "). Separator symbols and blank spaces cannot be used in names. Names cannot be longer than 32 characters. Names not included within quotation marks are not case sensitive i.e. "A" and "a" represent different names while A and a represent the same name.

Chip Names

Chip names can be composed of any allowed characters except separator characters and blank spaces. Chip names may be enclosed in quotation marks (" "). The same rules apply for pin names.

MOBIC Model Structure

The MOBIC source text can describe more than one model. The model structure is very rigid (i.e. statement sequence cannot be changed, nor can any statement be omitted). The model is composed of the following statements:

CIRCUITS list_of_chips;

a list of all chips described by the equations in the EQU section.

```
CLASS chip_class(es);  
    a list of chip class, e.g.. TTL, ECL  
  
PINS list_of_pins;  
    a list of chip pin names and numbers  
    equations or references to a model  
    already defined in the text.  
  
SIGNALS list_of_signals;  
    list all signals used within EQU;  
    an empty list is allowed (no signals used).  
  
EQU  
    enter logic equations describing the functional model behavior;  
END;
```

If you are using an existing model that already has defined SIGNALS and EQU statements, use the following statements:

```
CIRCUITS list_of_chips;  
PINS list_of_pins;  
USE existing_model_name;  
END;
```

As mentioned above, the source text may define more than one model:

```
CIRCUITS list;  
...  
END;  
  
CIRCUITS list;  
...  
END;
```

Reserved Words

Operators, constants and reserved words are case insensitive i.e. **END** and **end** represent the same word. The names listed below cannot be used as signal names. If a pin name is identical to any of the reserved words, the name must always be included in quotation marks. Statement and instruction names must always be followed by a blank space if arguments are present.

Statement Names

CIRCUITS	list of chip names described by equations in the EQU section
CLASS	class of chips
PINS	list of pins
SIGNALS	list of signals
EQU	beginning of equations section
END	end of the model
USE	points to an existing model; eliminates a need to redefine equations

Class Declarations

TTL	TTL voltage level on pins
ECL	ECL voltage level on pins
MOS	MOS voltage level on pins
ROM	ROM memory chip(s) - may be combined with TTL/ECL/MOS
RAM	RAM memory chip(s) - may be combined with TTL/ECL/MOS

Pin Types

IN	input pin
OUT	output pin (Totem Pole)
TRI3	3-state output pin
TRIIO	3-state bi-directional pin
OC	open collector output pin
OCIO	open collector bi-directional pin
OE	open emitter output pin
TTL	TTL pin voltage level (override class level), optional
MOS	MOS pin voltage level (override class level), optional
ECL	ECL pin voltage level (override class level), optional

Instruction Names

IF	conditional instruction
ELSE	case clause (optional)
ENDIF	end of conditional instruction
READ	read word from memory
WRITE	write word to memory
VIOLATION	report violation

Operators

Table 4-1. MOBIC Operators

IF instruction relational operators:	
EQ	equal
NE	not equal
Logic equation operators:	
=	assignment (put state to signal/pin)
*	AND
+	OR
#	XOR
-	NOT
()	parentheses

Logical States

Table 4-2 . Logical State List

Logic State	
LOW	Low
HIGH	High
Unkn_X	Unknown X
Hi_Z	High Impedance
Ref_V	Reference Voltage (ECL only)
Hi_V	High Voltage
Conf_X	Conflicting Signal States
Res_High	Resistive High
Res_Low	Resistive Low

Ua_Low	Unknown Activity Low
Ua_High	Unknown Activity High
Ua_X	Unknown Activity X

Power_On Signal

POWER_ON - this signal cannot be user defined. The **POWER_ON** signal can have a only logical HIGH or LOW state.

LOW - normal simulation,

HIGH - Power On process.

You can check the value of a **POWER_ON** signal if you want to set initial conditions in your model. The **POWER_ON** signal can be used in logic expressions like any other signal. However, the user cannot assign to it any value; this is a read-only signal.

The **POWER_ON** signal can also be used as an IF instruction argument. Because it may have only two logic states, a comparison with a state other than the LOW or HIGH logic state will be considered an error during compilation.

Example:

```
IF POWER_ON EQ HIGH
    Clear = HIGH
ENDIF;
```

Violation Codes

Any code used with VHDL is legal but most codes should not be used because they report timing violations (MOBIC models are functional). Recommended codes are as follows:

V_UN_IN	undetermined input pin state
V_UN_CLK	undetectable clock pulse
V_LOG	LOGIC error in cycle (for DRAM)
V_ABORT	undetermined pin state - CYCLE ABORTED (for DRAM)

Statement Syntax

All statements can be written in one or more text lines. The semicolon (;) character indicates the end of a statement. Blank spaces can be inserted between words and separators as needed.

CIRCUITS

CIRCUITS **name1, name2, ... , nameN** ; (unlimited number of names)

wherein:

- , is the name separator
- ; always terminates statement

The above CIRCUIT statement may also be written as follows:

```
CIRCUITS name1,  
          name2,  
          ...  
          nameN;
```

Example:

```
CIRCUITS 7400, 5400,  
          74LS00, 54LS00;
```

CLASS

CLASS **type , ram/rom:0/1** ;

where:

type - TTL, MOS or ECL specifies default voltage level of all pins

ram/rom:0/1 - optional RAM or ROM class:

- 0/1 - virgin state (optional argument, default setup is 0)
- : - virgin state argument separator
- ram/rom - RAM or ROM class declaration

Examples:

```
CLASS ECL;  
CLASS TTL, RAM;
```


CLASS TTL, ROM:1; \ default ROM state is 1, unless overwritten with a hex file.

PINS

PINS name1 :type : volt_level :number , ...;

where:

name1 - pin name (must be present)
type - type definition with separator (:)
volt_level - voltage level - overrides class level [optional]
number - pin number with separator (:) [optional]

Pin number is an integer up to 3 digits (e.g.. 1, 15) or a letter with a 1 or 2-digit number (e.g.. A1, B12)

Examples:

PINS A1:IN:1, B1:IN:2, Y1:OUT:3;
PINS A1:IN:TTL:1, A2:IN:2, Y:OE:15; - example shows voltage override declaration (TTL)
PINS IN1:IN, IN2:IN, OUT:OUT; - no pin numbers, name OUT is also a reserved word.

SIGNALS

SIGNALS sign1, sign2, ... , signN ;

The **SIGNALS** statement syntax is identical to the **CIRCUITS** statement but the signal list may be empty. The **SIGNALS** statement can be followed only by the **EQU** statement.

Examples:

SIGNALS A1, btp; \ A1 and btp are used in the equation
SIGNALS ; \ empty list; no signals are used within EQU

EQU

EQU - no arguments, ; cannot be present

END

END ; - no arguments, ; must be present

USE

USE chip_name (; cannot be present)

where:

chip_name is the name of the device defined by an existing model.

This statement is useful for defining a chip when the model is already defined but has a different pin layout.

Example:

```
CIRCUITS      7400, 5400;
CLASS        TTL;
PINS         A2:IN:4, B2:IN:5, Y2:OUT:6, ...;
SIGNALS tmp;
EQU
    \ logic equations
END;
CIRCUITS      7400W, 5400W;          \ the same model as above, dif-
ferent pin layout
CLASS        TTL;
PINS         A2:IN:6, B2:IN:7, Y2:OUT:5, ...;
USE          7400 - \ note the omitted SIGNALS and EQU
END;
```

Note: Pin names and types in the model specified within the USE statement must be exactly the same as those listed in the current model. The CLASS statements must also be identical; the MOBIC compiler will check for this.

Instruction Syntax

Instructions can be written in one or more text lines. The semicolon (;) indicates the end of the instruction. Blank spaces can be inserted between words and separators, as needed. An indentation can also be used.

Conditional Instruction IF

```
IF signal/pin_name operator constant
    \ other instructions or equations
ELSE
    other instructions or equations (optional)
ENDIF;
```

where:

signal/pin_name - name of signal or pin. The state of this signal/pin will be compared with a CONSTANT. The special POWER_ON can also be used.

operator - one of two relational operators: **EQ** (equal) or **NE** (not equal)

constant - a logic state

An **IF** instruction can be nested inside another **IF** instruction. The number of levels is not limited.

Example:

```
IF POWER_ON EQ HIGH
    Y = UNKN_X;
ELSE
    IF A NE Hi_Z
        Y = A*-B;
    ELSE
        IF B EQ Hi_Z
            Y = UNKN_X;
        ENDIF;
    ENDIF;
ENDIF;
```

Read from Memory (Instruction READ)

READ (addr0, ..., addrN), (data0, ..., dataK);

where:

(addr0, ..., addrN) - is an address signals list: **addr0** is the least significant bit (LSB) and **addrN** is the most significant bit (MSB); parentheses must be used

(data0, ..., dataK) - is a data signals list: **data0** is the LSB and **dataK** is the MSB; parentheses must be used

Pin names and temporary signals may be used in both address and data lists. The READ instruction automatically generates a VIOLATION re-

port if any of the address list elements is in an undetermined state (e.g. HI_Z, UNKN_X, etc.). It assigns the UNKN_X state to the addressed data (signal) on the data list.

Example:

16x4-bit ROM chip is modeled as follows.

```

CIRCUITS      ROM4x16;
CLASS        TTL:ROM:1;
PINS         ADDR0:IN, ADDR1:IN, ADDR2:IN, ADDR3:IN,
D0:TRI,D1:TRI,D2:TRI,D3:TRI, \ CS:IN;
SIGNALS      ;
EQU
    IF \ CS EQ LOW
        READ (ADDR0, ADDR1, ADDR2, ADDR3), (D0, D1,
D2, D3);
    ELSE
        D0 = HI_Z;
        D1 = HI_Z;
        D2 = HI_Z;
        D3 = HI_Z;
    ENDIF;
END;

```

Write to Memory (Instruction WRITE)

WRITE (addr0, ..., addrN), (data0, ..., dataK); This instruction stores **data0...dataK** at the **addr0...addrN**.

where:

(addr0, ..., addrN) - is the address signals list (addr0 = LSB addrN = MSB)
parentheses must be used

(data0, ..., dataK) - is the data signals list (data0 = LSB dataK = MSB)
parentheses must be used

Pin names and temporary signals may be used in both the address and data lists. The WRITE instruction automatically generates a VIOLATION report if any address or data list element is in an undetermined state (e.g. HI_Z, UNKN_X, etc.). Memory contents, however, are not updated.

Example:

The 16x4-bit RAM chip is modeled as follows:

```

CIRCUITS      RAM4x16;
CLASS        TTL:RAM;
PINS         ADDR0:IN, ADDR1:IN, ADDR2:IN, ADDR3:IN,
D0:TRIIIO,D1:TRIIIO, D2:TRIIIO,D3:TRIIIO, \ CS:IN,

```

```
\ WE:IN;
SIGNALS ;
EQU
IF \ CS EQ LOW
    IF \ WE EQ HIGH
        READ (ADDR0, ADDR1, ADDR2, ADDR3), (D0, D1, D2,
D3);
    ELSE
        WRITE (ADDR0, ADDR1, ADDR2, ADDR3), (D0, D1,
D2, D3);
    ENDIF;
ELSE
    D0 = HI_Z;
    D1 = HI_Z;
    D2 = HI_Z;
    D3 = HI_Z;
ENDIF;
END;
```

Report Violation (Instruction VIOLATION)

VIOLATION **code** , **pin_name** ;

where:

code is the violation code

pin_name is the name of the faulty pin; both pin name and separator (.) are optional. If the faulty pin cannot be determined, specify only the violation code

Example:

```
Clk_Pulse = -Old_Clk * CLK;
IF Clk_Pulse EQ HIGH
    IF Din EQ UNKN_X
        VIOLATION V_UN_IN, D;
    ELSE
        Q = Din;
    ENDIF;
ENDIF;
```

Logical Equation Syntax

Equation Syntax

Logic equations are written in the same way as are standard Boolean equations. There are 12 logic states but you don't have to worry what they mean, because they are processed automatically by the MOBIC compiler and the ACTIVE-CAD simulator. Spaces can be added anywhere between names, constants and operators. The equation can also be written on more than one line.

Each logic equation is comprised of the following elements:

- ☐ left-hand side is an operand which can only be a pin or signal
- ☐ assignment operator (=)
- ☐ right-hand side is a logic expression or constant
- ☐ terminator character ;

Examples:

```
Y = High;  
X = -(A1*B + C)#D;
```

Operator priority

Operators are listed below in order of precedence, the highest precedence (priority) is listed first:

- () - parentheses
- - NOT
- * - AND
- # - XOR
- + - OR

12-Value Logic

Not all logic states can be produced by logic expressions. Only the following logic states may be produced by logic expressions:

LOW,
HIGH,
Unkn_X.

These logical states can be assigned to output pins and signals.

The following states can be assigned to any output pin using the assignment operator (=):

Hi_Z,
Ref_V,
Hi_V,
Res_High,
Res_Low,
Ua_Low,
Ua_High,
Ua_X,

You can also use the above listed states as IF statement conditions (e.g. IF sig=Res_High THEN...). These states can never be the result of a logic expression or assigned to internal signals.

The Conf_X state (bus conflict) cannot be generated by a model at all. This is a special state created by the netlist analysis procedure, when a BUS conflict occurs. At run-time MOBIC always treats this state as Unkn_X, so the user does not have to check for it. Because of that, this state should not be listed in the truth tables.

Res_High and Res_Low states are logically equivalent to HIGH and LOW states respectively and should not be listed in the truth tables.

Ref_V and Hi_V are special-purpose states and should not be assigned to any pins used in the logic expressions. The user should check for them first and use the IF instruction to process these states (e.g. report a viola-

Table 4-3. Truth Table for NOT Operation.

A	- A
LOW	HIGH
HIGH	LOW
Unkn_X	Unkn_X
Hi_Z	Unkn_X
Ref_V	Unkn_X
Hi_V	Unkn_X
Conf_X	Unkn_X
Res_High	LOW
Res_Low	HIGH
Ua_Low	Unkn_X

Table 4-4. Truth Table for AND Operation

B	A	B*A
LOW	LOW	LOW
LOW	HIGH	LOW
HIGH	LOW	LOW
HIGH	HIGH	HIGH
Unkn_X	LOW	LOW
Unkn_X	HIGH	Unkn_X
Unkn_X	Unkn_X	Unkn_X
Unkn_X	Hi_Z	Unkn_X
Hi_Z	LOW	LOW
Hi_Z	HIGH	Unkn_X
Hi_Z	Unkn_X	Unkn_X

tion). The Ref_V is an ECL voltage level used to convert a differential input pair to a single-ended input. The Hi_V level is a TTL high voltage level (above +5V) which can be used to simulate PROM programming. Ua_Low, Ua_High, and Ua_X are assigned to 3-state outputs when it is

Table 4-5. Truth Table for
OR Operation

B	A	B+A
LOW	LOW	LOW
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	HIGH
Unkn_X	LOW	Unkn_X
Unkn_X	HIGH	HIGH
Unkn_X	Unkn_X	Unkn_X
Unkn_X	Hi_Z	Unkn_X
Hi_Z	LOW	Unkn_X
Hi_Z	HIGH	HIGH
Hi_Z	Unkn_X	Unkn_X
Hi_Z	Hi_Z	Unkn_X

Table 4-6. Truth Table for
XOR Operation

B	A	B#A
LOW	LOW	LOW
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	LOW
Unkn_X	LOW	Unkn_X
Unkn_X	HIGH	Unkn_X
Unkn_X	Unkn_X	Unkn_X
Unkn_X	Hi_Z	Unkn_X
Hi_Z	LOW	Unkn_X
Hi_Z	HIGH	Unkn_X
Hi_Z	Unkn_X	Unkn_X
Hi_Z	Hi_Z	Unkn_X

impossible to determine if the output should be in a strong or High Impedance state. The MOBIC compiler always assumes the worst case and processes these states as Unkn_X in all logic expressions.

The truth tables of all logic operations are listed in tables 4-3 through 4-6. The **NOT** operation truth table includes all 12 logic states. The other tables are simplified and list only 4 logic states, accordingly to the above explanations.

Comments

There are two types of comments in the MOBIC Language:

line comment - starts with \ (backslash) character and ends at the end of the line;

block comment - starts and ends with the double backslash \\
and may include many lines.

Example:

```
Y = A + C; \ this is a line comment
  \ \      This is a block comment, all lines below are comments
        until the next double backslash appears;
        \ any line comments inside block comment remain unaffected
  \ \
```

Note: For text clarity the user can insert empty lines into his/her program.

Signals and Pins

There are two types of logic variables used in the MOBIC language:

pins - defined within the **PINS** statement; define variables through which the model communicates with the simulator and other models;

signals - other variables used in logic equations are defined within the **SIGNALS** statement. If no signals are used in equations, the **SIGNALS** statement includes an empty list.

Memory Model Addressing

Your model can store up to 16k words. Examples: memory 256K x 8 will be modeled as 16K x 8 chip, memory 64K x 1 will be modeled as 16K x 1, chip 8K x 1 will be the same 8K x 1. Addresses overlap at the 16K boundary to save simulator memory. You can model 1, 2, ... ,64-bit memories.

Sample Models

Gates

```

CIRCUITS 7400,5400;          \ Quad 2-input NAND
CLASS TTL;
PINS 1A:IN:1, 1B:IN:2, 1Y:OUT:3,
      2A:IN:4, 2B:IN:5, 2Y:OUT:6,
      3A:IN:9, 3B:IN:10,3Y:OUT:8,
      4A:IN:12,4B:IN:13,4Y:OUT:11;
SIGNALS ;
EQU
1Y = -(1A*1B);
2Y = -(2A*2B);
3Y = -(3A*3B);
4Y = -(4A*4B);
END;
CIRCUITS 7400W,5400W;        \ Quad 2-input NAND
ceramic flat package
CLASS TTL;
PINS 1A:IN:1, 1B:IN:2, 1Y:OUT:3,
      2A:IN:6, 2B:IN:7, 2Y:OUT:5,
      3A:IN:9, 3B:IN:10,3Y:OUT:8,
      4A:IN:12,4B:IN:13,4Y:OUT:14;
USE 7400
END;

```

Multiplexer

```

CIRCUITS 74152,74LS152;      \ 1-of-8 data selec-
tors/multiplexers
CLASS TTL;
PINS
D0:IN:5,D1:IN:4,D2:IN:3,D3:IN:2,D4:IN:1,D5:IN:13,D6:IN:12
,
      D7:IN:11, A:IN:10, B:IN:9, C:IN:8, W:OUT:6;
SIGNALS ;
EQU
W = -(-A*-B*-C*D0 +
      A*-B*-C*D1 +
      -A* B*-C*D2 +
      A* B*-C*D3 +
      -A*-B* C*D4 +
      A*-B* C*D5 +
      -A* B* C*D6 +
      A* B* C*D7);
END;

```

D Flip-Flop

```

CIRCUITS 7474;                                \ Dual D-type flip-
flops
CLASS TTL;
PINS CLR1:IN:1, D1:IN:2, CK1:IN:3, PR1:IN:4,
Q1:OUT:5,/Q1:OUT:6,
CLR2:IN:13,D2:IN:12,CK2:IN:11,PR2:IN:10,Q2:OUT:9,/Q2:OUT:
8;
SIGNALS tmp, clk, OldCK1, OldCK2, Din1, Din2;
EQU
IF POWER_ON EQ HIGH                          \ Power ON Procedure
    tmp = CLR1 * PR1;                        \ Section 1
    IF tmp EQ HIGH
        Q1 = UNKN_X;
        /Q1 = UNKN_X;
    ELSE
        tmp = CLR1 # PR1;
        IF tmp EQ UNKN_X
            Q1 = UNKN_X;
            /Q1 = UNKN_X;
        ELSE
            Q1 = CLR1 + -PR1;
            /Q1 = -Q1 + -CLR1;
        ENDIF;
    ENDIF;
tmp = CLR2 * PR2;                            \ Section 2
IF tmp EQ HIGH
    Q2 = UNKN_X;
    /Q2 = UNKN_X;
ELSE
    tmp = CLR2 # PR2;
    IF tmp EQ UNKN_X
        Q2 = UNKN_X;
        /Q2 = UNKN_X;
    ELSE
        Q2 = CLR2 + -PR2;
        /Q2 = -Q2 + -CLR2;
    ENDIF;
ENDIF;
ELSE                                          \ Normal Operation
    tmp = CLR1 # PR1;                        \ Section 1
    clk = -OldCK1 * CK1;
    IF tmp EQ UNKN_X
        Q1 = UNKN_X;
        /Q1 = UNKN_X;
    ELSE
        Q1 = (-clk*Q1 + clk*Din1) * CLR1 + -
PR1;
        /Q1 = -Q1 + -CLR1;

```

```

ENDIF;
tmp = CLR2 # PR2;          \ Section 2
clk = -OldCK2 * CK2;
IF tmp EQ UNKN_X
    Q2 = UNKN_X;
    /Q2 = UNKN_X;
ELSE
    Q2 = (-clk*Q2 + clk*Din2) * CLR2 + -
PR2;
    /Q2 = -Q2 + -CLR2;
ENDIF;
OldCK1 = CK1;              \ Update old states
Din1  = D1;
OldCK2 = CK2;
Din2  = D2;
ENDIF;
END;

```

Magnitude Comparator

```

CIRCUITS 7485;              \ 4-bit Magnitude
comparator
CLASS TTL;
PINS A0:IN:10, A1:IN:12, A2:IN:13, A3:IN:15,
      B0:IN:9,  B1:IN:11, B2:IN:14, B3:IN:1,
      IN1:IN:2, IN2:IN:3, IN3:IN:4, OUT1:OUT:5,
OUT2:OUT:6, OUT3:OUT:7;
SIGNALS n;
EQU
n = A0#B0 + A1#B1 + A2#B2 + A3#B3;
IF n EQ HIGH
    OUT1=A3*-B3+-(A3#B3)*(A2*-B2+-(A2#B2)*(A1*-
B1+-(A1#B1)*A0*-B0));
    OUT3 = -OUT1;
    OUT2 = A3*-A3;
ELSE
    OUT1 = -IN2*-IN1;
    OUT3 = -IN2*-IN3;
    OUT2 = IN2;
ENDIF;
END;

```

EPROM Memory

```

CIRCUITS 2764A;             \ 8Kx8 UV ERASABLE
FROM
CLASS TTL,ROM:1;

```

```
        PINS A0:IN:10, A1:IN:9, A2:IN:8, A3:IN:7,
A4:IN:6, A5:IN:5, A6:IN:4,
            A7:IN:3, A8:IN:25, A9:IN:24, A10:IN:21,
A11:IN:23, A12:IN:2,
            O0:TRI:11, O1:TRI:12, O2:TRI:13, O3:TRI:15,
            O4:TRI:16, O5:TRI:17, O6:TRI:18, O7:TRI:19,
            OE:IN:22, CE:IN:20;
SIGNALS n;
EQU
n = OE + CE;
IF n EQ HIGH
    O0 = HI_Z;
    O1 = HI_Z;
    O2 = HI_Z;
    O3 = HI_Z;
    O4 = HI_Z;
    O5 = HI_Z;
    O6 = HI_Z;
    O7 = HI_Z;
        Violation V_lead,a1;
ELSE
    READ
(A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12),
        (O0,O1,O2,O3,O4,O5,O6,O7);
ENDIF;
END;
```

Chapter 5

Simulator Macro Operations

1. GENERAL

1.1 Simulator macros

Simulator macros allow you to specify a sequence of operations to be performed by the simulator on a design. You can specify the design configuration, test vectors to be applied to the design, test points to be recorded, and display conditions. The macro operations are a convenient way to run design simulations in batch mode and collect vast amounts of design data off-line. The simulator macro can be used for the final ASIC design verification, system level design verification, ECO (Engineering Change Order) verification and approval, automatic parts selection and replacement, and many other applications.

NOTE 1: The waveforms or test vectors that are generated by the macro are weak signals, which means that they can be overridden by strong signals on the device outputs and in the nodes driven by strong signals. However, when applied to device inputs, they cannot be overridden by any other signal.

ALDECs macro are similar to ViewLogics ViewSim commands. For better compatibility, the ALDEC macro use the same command syntax as ViewSim.

NOTE 2: The simulator macro are available only with the ACTIVE-CAD product lines. They are not available with the SUSIE-CAD products.

1.2 Naming conventions

COMPONENT NAMES

Simulator macro allow you to use any schematic or netlist component names. The hierarchical component location can be specified by using similar to a DOS path name, with the top hierarchical level being listed first. The root or top-most hierarchical level can be specified as:

root/U1 chip U1 at the root level

or U1 chip U1 at the root level

Hierarchical levels are separated by slashes (/), e.g:

B11/B12/U22

U22 is located at the B12 hierarchical level which is a sub-hierarchy of B11.

You cannot use wild characters for component search. They need to be listed precisely.

PIN NAMES

macro accept any component pin names. The pin names are specified by listing the component name, followed by a period (.) separator and pin name. The hierarchical pin name location can be specified using a path, with the top hierarchical level being listed first. The root or top-most hierarchical level signal can be specified as:

root/U1.A1 pin U1.A1 at the root level
or U1.A1 pin U1.A1 at the root level

Hierarchical levels are separated by slashes (/), e.g.

B11/B12/U25.Y1

U25.Y1 pin is located at the B12 hierarchical level, which is a sub-hierarchy of B11.

You cannot use a wild character for pin name search. It needs to be listed precisely.

SIGNAL NAMES

macro accept any schematic and netlist signal names. The hierarchical signal location can be specified by means of path, with the top hierarchical level being listed first. The root or top-most hierarchical level signal can be specified as:

root/RESET signal RESET at the root level

or RESET signal RESET at the root level

Hierarchical levels are separated by slashes (/), e.g:

B11/B12/ENABLE

The ENABLE signal is located at the B12 hierarchical level which is a sub-hierarchy of B11.

You cannot use wild characters for component search. They need to be listed precisely.

1.3 Logical states

Logical states are represented by names (HIGH, LOW, etc.) and by numbers (0,1). Bus logical states are described by a list or an array of names or numbers. The left-most names or numbers represent the highest bus signal values. If you specify fewer bus names than allowed, macro will automatically fill in 0s or Lows for the higher order bus lines. A single signal line is treated as a bus with a single line.

Buses can be expressed in Binary (\B), Octal (\O), Decimal (\D) and Hexadecimal (\H) format. If no format is specified, it is assumed to be a binary. These formats may also be expressed using their numerical bases, e.g. \2, \8, \16, \20.

Examples of buses:

<input type="checkbox"/> 11100010101	binary bus representation
<input type="checkbox"/> 100010101\B	binary bus representation
<input type="checkbox"/> 1234\8	octal bus representation
<input type="checkbox"/> 777\O	octal bus representation
<input type="checkbox"/> eff3ab\16	hexadecimal bus representation
<input type="checkbox"/> 10010101\H	hexadecimal bus representation
<input type="checkbox"/> 99999\D	decimal bus representation
<input type="checkbox"/> abcdefg\20	base 20 numbering system

If you specify a bus signal state, as shown below, the signal state pertains to the entire bus (all its signal lines will have the same signal state):

BUS Signal State	Description
------------------	-------------

Z	HI_Z
RL	RES_LOW
RH	RES_HIGH
RX	RES_X
L	LOW
H	HIGH
X	UNKN_X
UL	UA_LOW
UH	A_HIGH
CX	CONF_X
R	REF_V
V	HIGH_V
SL	SV_LOW
SV	SV_HIGH
SX	SV_X

1.4 Time units

Time is specified in decimal values. Its units are listed below. The default ACTIVE-CAD time unit is 10 picoseconds, and all simulations are made with this precision, . However, for full compatibility with ViewLogic, the default time unit used in all macro is 100 picoseconds. If you want to specify timing with a 10 picosecond resolution, you must write ps explicitly. If you omit the units, time will automatically be counted in 100 picosecond units.

<input type="checkbox"/> ps	10 exp (-12) seconds
<input type="checkbox"/> ns	10 exp (-9) seconds
<input type="checkbox"/> us	10 exp (-6) seconds
<input type="checkbox"/> ms	10 exp (-3) seconds
<input type="checkbox"/> s	seconds
<input type="checkbox"/> m	minutes
<input type="checkbox"/> h	hours

Time can be referenced to past events or to the beginning of the simulation. If time is referenced to the beginning of the simulation, then the time values are preceded by the @ marker.

Examples:

```
100 - indicates 10ns (100x100ps) from the last event for
the selected signal line
70ps - indicates 70ps
@153us indicates 153 microseconds from the beginning of
the simulation
```

Note: The units must be placed right after the number. For example, 120ns is correct. However, 120 ns is incorrect because the space is treated as a delimiter, and the ns marker will not be recognized by the software.

Simulation precision is set manually. For compatibility with ViewSim, it cannot be set from a macro. Only timing precision can be set by using macro operations. For example, if the simulation precision is set to 10ps and the timings resolution is 100ps, then the simulator will accept the timing with 100ps resolution but will generate output signals with 10ps resolution. On the other hand, if the timing is given with 10ps resolution but the simulation precision has been set to 1ns resolution, then the ACTIVE-CAD simulator will round off the timing to the nearest 1ns.

2. Macro File Structure

You can enter and edit the macro operations file under any ASCII text editor, such as Notepad and others. macro are separated by semicolons (;) and [ENTER]. If a macro does not fit in a single line, you need to end the line with a plus (+) sign and continue the macro in the next line.

Instead of the full macro names, you can use their special abbreviations, which are listed with each macro description.

macro are executed in the sequence listed in the file, except when there are loops. The execution of the macro is ended when:

- ☐ the last macro in the file has been executed
- ☐ the program has encountered the quit macro
- ☐ the simulator has encountered a fatal error that inhibits further processing of the macro.

The macro operations file should have a .CMD file name extension. The macro file should be added to your project under the **Project Manager** as a project resource. Follow these steps to add macro file to your project:

- ☐ Select **Resources** from the **Project Manager**
- ☐ From the **Directories** field (in the **Project Resources** window) select the directory that contains the macro files
- ☐ From the **Resource Type** field select the **Stimulator macro**
- ☐ When the **Additional resources available in directory...** window displays the desired macro file, click on the file and then select either the **Copy** or **Link** option.

The macro added to the current project can then be executed by selecting in sequence:

Utilities Macro Run operations

If the macro file uses additional data files, they need to be added to the project **before** invoking the **Run** operation. The default directory (catalog) for these files is the Project directory.

The currently executing macro is listed in the simulator status window.

3. Test Vectors

The simulation process is comprised of two steps:

1. Create a data file with test vector values.
2. Write a command file that loads (assigns) test vector values and issues the simulate command to simulate the loaded values.

3.1. Test vector files

Each line in a test vector file lists a set of logical values to be assigned to a signal line or a bus at a selected time instance. Comments lines start with the symbol |.

Test Vector File Example:

```
| ab.dat test vector file
| test vectors for buses A i B
|
| format
| A[0:7]
| B[0:3]
|
10101101 | binary value
2\D      | decimal value of 2; uses four lines B[0:3]
|
AB\H     | hexadecimal value of AB
0111
|
10001111
Z
|
```

In the above example, only the lines without the | symbol will be loaded into test vector files (executed by the macro processor).

Command File Example:

```

Assign A ab.dat | assigns the first data (10101101) from ab.dat to the
A bus
    Assign B ab.dat | assigns the second data (2\D) from ab.dat to the B
bus
    sim           | simulates the assigned bus values
Assign A ab.dat  (AB\H) | assigns the 3rd data from ab.dat file
Assign B ab.dat  (0111) | assigns the 4th data from ab.dat file
sim
Assign A ab.dat  (10001111)
Assign B ab.dat  (Z)
sim
| The Assign macro automatically loads the consecutive data from the ab.
dat file.

```

3.2. Event Files

Each line in the event file describes the new signal or bus state and the time of its occurrence from the beginning of the simulation.

Event File Example:

```

@0 = 0\H (hexadecimal)
@100 = 1\H
@200 = Z
@300 = A\H

```

4. Viewsim Compatible macros**4.1. Comments**

Format: |

Abbreviation: none

Operation

A line starting with the | comment symbol is omitted during macro execution. However, the text of the comments is displayed in the simulators comment window.

Comment Example:

```
| this is a procedure for testing the decoder
```

4.2. Macro (Command) Loops

Format: `(list_of_commands_in_the_loop)*n`

Abbreviation: none

Operation

The macro (commands) listed in parentheses are performed n times, where n is a decimal number. A loop within a loop is not allowed.

Example:

```
(assign A a.dat; assign B b.dat; sim; print results.dat) * 12
```

This macro will repeat 12 times the following operations:

- ☐ assign to the signal (bus) A the next value listed in the *a.dat* file
- ☐ assign to the signal (bus) B the next value listed in the *b.dat* file
- ☐ simulate one simulation step
- ☐ write the signals listed in the **Watch** file to the *results.dat* file.
- ☐ repeat the first operation, etc.

4.3. Activity

Not implemented

4.4. After

Format: after time do (cmd1;cmd2;...cmdn)

Abbreviation: none

Operation:

AFTER defers the execution of the set of commands listed in parentheses until after the specified time has elapsed, relative to the current simulation time. The commands in each macro execute in the order in which they are listed. However, if more than one AFTER macro is scheduled to execute at the same time, the order of execution of the two macros is undefined.

The commands must each be separated by a semi-colon.

Example:

```
after 10ns do (assign DATA0 1)
sim
```

After 10ns of simulation (relative to the current time), the signal DATA0 assumes a constant value of 1.

4.5. Assign

Format: **assign** *signal_or_bus* *new_state*
 assign *signal_or_bus* *data_file*

Abbreviation: none

Operation

The second parameter is assigned to the first one during the simulation. The second parameter can be either a concrete signal state or a signal value from the specified file. The assigned signal behaves as a weak signal, and the node is controlled by the actual chip outputs, if any.

Example:

```
assign U1.A1 Z      | assigns a tri-state to the U1.A1 pin
assign ADBUS1 AB03\H | assigns hexadecimal value AB03
                     to the bus ADBUS1
assign DATA data_bus.dat | assigns to DATA the values
                           from the data_bus.dat file
```

4.6. Breakpoint

Format: **break**
 break *signal_or_test_vector*
 break *signal_or_test_vector* *event*
 break *signal_or_test_vector* *event* *do* (sequence of events)

Abbreviation: break

Operation

This macro clears breakpoints as a result of aspecified simulation result (event). It also stops simulation when specified signal or test vector conditions are met.

- ☐ **break** ; clears all breakpoints
- ☐ **break *signal_or_test_vector*** ; clears all breakpoints set for specified signal, device pin or test vector
- ☐ **break *signal_or_test_vector event*** ; stops the simulation process if the breakpoint condition (*event*) is met. Executes the next macro from the macro file. If the breakpoint was a part of a loop, the loop is aborted and the next macro is executed.
- ☐ **break *signal_or_test_vector event do*** (sequence of events) ; stops the simulation when the selected breakpoint condition is met, and then executes the macro listed after *do*. After the completion of these macro, ACTIVE-CAD continues to execute the macros from the macro file.

Event format:

- ☐ ? - means that any change is the desired event
- ☐ condition - indicates the desired state
- ☐ condition1 - condition2 - the desired state is achieved upon changing from *condition1* to *condition2*

Examples:

break - deletes (turns off) all breakpoints

break clock - deletes all breakpoints on clock signal

break clock ? - stops simulation after any clock transition

break enable Z-X - stops simulation when the enable signal changes from High_Z to Unknown_X

break clock 0-1 do (assign DATAfile.dat; print) - stops simulation when the clock transitions from low to High and then assigns signal DATA a new value from file.dat. Next, it prints the new state of all signals.

4.7. Changes

Not implemented

4.8. CHECK

Format: `check signal_name [> filename]`

`check signal_name [value]`

Abbreviation: none

Operation:

With no options, this command writes the value of the signal to the console. If you specify a filename, then the command writes the value to the file.

If you specify a value, then the command compares the signals value to the value specified. If there is a discrepancy between the two values, the command writes a message to the console. Instead of specifying the value directly, you may read in values from a file. Each time you enter the command, it reads in one value from the file, then advances the file pointer to the next value.

Examples:

DATA0 is a signal whose value at 100ns (the current time) is 1.

`check DATA0`

DATA0 = 1 at time 100ns

The command writes the current value of DATA0 to the console.

`check DATA0 >out`

The command writes the current value of DATA0 to file out in the current directory.

`check DATA0 1`

The value of DATA0 at the current time equals the value specified, so the command does not write a result.

`check DATA0 0`

value 1 on DATA0 does not match 0 at time 100ns

The command writes to the console the fact that the value of DATA0 at the current time does not match the value specified.

File in contains values 0,1,0,1.

```
check DATA0 <in
value 1 on DATA0 does not match <in at time 100ns
check DATA0 <in
check DATA0 <in
value 1 on DATA0 does not match <in at time 100ns
check DATA0 <in
```

Each time the check command is executed with the <in parameter, the next value is read in. When the end of the in file is reached, the file re-winds.

4.9. Clock

Format **clock** *signal_or_bus_specification* *signal_state_1 ... signal_state_n*

Abbreviation **ck**

Operation

The **clock** macro defines a periodical signal or bus which is applied to the node. Each signal state lasts for the duration of the SHORT step. By setting the STEP to different values, different clock timings can be generated.

Clock causes automatic synchronization of all signals and buses generated by the SETUP operation. Each new signal state, as defined by the SETUP operation, is applied to the simulator at the beginning of the clock cycle. If the clock cycle is comprised of four simulation steps (or logical states), then the SETUP signals will be applied every four simulation STEPS.

Example:

```
step 20ns
clock c 0 1 0 1 1 0 0 1
pattern b 0 1
pattern a 0
run
```

4.10. Continue

Not implemented

4.11. Cycle

Format: **cycle**
 cycle *n*

Abbreviation **c**

Operation

The simulator performs an *n* number of clock cycles, which need to be defined prior to using this macro (see SETUP Clock macro). If no clock has been specified, the SHORT step is used. If several clocks have been defined, the longest clock period is used. If no *n* is listed, only one simulation cycle is performed.

Example:

```
Cycle 5  
c 5
```

These macro will simulate five clock cycles that have been defined by the SETUP Clock macro.

4.12. Defaults

Not implemented

4.13. DELAY

Not implemented

4.14. DISPLAY

Format: display

Abbreviation: d

Operation:

This macro displays the values of signals which have been previously selected with the WATCH command. If no signals were selected, then the macro returns the current simulation time.

Examples:

```
display ; no signals were selected previously
time = 50ns ; display current time only
watch DATA0 ; select DATA0
display
time = 50ns DATA0 = 0 ; display current time and
                        value of DATA0 at that time
```

4.15. DUMPM

Not implemented

4.16. ECHO

Format: **echo** *any text free of semicolon and new lines*

Abbreviation none

Operation

This macro writes into the simulators **Interact** window the text listed in the macro. If the macro has been used in a loop, the text cannot include a closing bracket. The macro is used to provide information about the simulators internal states and about the simulated circuit.

Example:

```
echo Start execution of memory test macro
```

4.17. EVERY

Format: every time do (cmd1;cmd2;...cmdn)

Abbreviation: none

Operation:

The commands in parentheses execute in order at time intervals specified in the time parameter. In order for this macro to take effect, it must precede a SIM, RUN, or CYCLE command.

If more than one of these macros specifies that commands should execute at the same time, the order of execution of the macros is undefined.

Examples:

```
w DATA0 ; select a signal whose values will be displayed
every 10ns do (d)
sim
time = 50ns DATA0 = 0
time = 60ns DATA0 = 1
time = 70ns DATA0 = 0
time = 80ns DATA0 = 1
time = 90ns DATA0 = 0
time = 100ns DATA0 = 1
```

4.18. EXECUTE

Format: execute command_filename

Abbreviation: ex

Operation: This command executes the commands stored in the command_filename.

Example:

```
ex cmd.fil ; the commands in cmd.fil are executed
```

4.19. FLUSH

Not implemented

4.20. High

Format: **high** signal_or_bus_specification

Abbreviation: **h**

Operation

This macro forces a weak High signal state on the specified signal or bus lines (by *signal_or_bus_specification*). If applied to an input pin, it will force that input pin to logical High. However, since this is a weak signal, if it is applied to an output pin or signal node, it will be overridden by any strong signal generated by the output pin or node.

Example:

```
h U21.INPUT ;
```

It forces a weak High signal state on the U21.INPUT pin. Since input override signals are never overridden by node signals, this weak signal will directly control the input pin.

4.21. Info

Not implemented

4.22. Inputs

Not implemented

4.23. Low

Format: **low** signal or bus specification

Abbreviation: **l**

Operation

The listed signal or bus will be forced into a weak Low signal state. If the macro is applied to an input pin, it will force that input pin to logical Low. However, if the macro is applied to an output pin or signal node, the signal will be overridden by any strong signal generated by the output pin or node.

Example:

```
low U12.OUTPUT ;
```

This macro forces a weak Low signal state into the U12.OUTPUT pin. This signal will be overridden by the chips strong output signal, if present.

4.24. Logfile

Not implemented

4.24. LOG_VECTORS

Not implemented

4.25. NETWORK

Format: network filename (no extension)

Abbreviation: net

Operation: This macro is equivalent to ACTIVE-CADs File/Load Netlist command.

Example:

```
net dacdemo ;load the dacdemo netlist into the simulator
```

4.26. Path

Not implemented

4.27. Pattern

Format: **pattern** *signal_or_bus_listing explicit_signal_state_listing*
 pattern *signal_or_bus_listing file_signal_state_listing*

Abbreviation: **pat**

Operation:

The specified signals and buses are forced into the explicitly listed signal states, or states listed in the file, in the listed sequence. The signals are forced into weak states, which means that they will control only input pins. However, if the macro is applied to output pins or nodes, the signal states may be overridden by strong signals. Each new signal state is applied at the beginning of the clock cycle. If no clock has been specified, each new signals state is applied at the beginning of each new simulation cycle (Step).

Example:

```
pattern U23.Y2 0 1 AB\H Z X H 12\D
pat B34.CLK .dat
```

This macro will assign to the pin U23.Y2, at the successive simulation cycles, the following signals:

L H H (lowest order bit of AB/h=H) High_Z Unkn_X L (lowest order bit of 12\D=0). Had the macro listed a bus in place of the single-wire pin U23.Y2, then the AB\H and 12\D would have been used as multi-signal (wire) data.

Example:

```
pat B34.CLK .dat
```

This macro will assign to the B34.CLK pin, at the successive simulation cycles, the signals from the clock.dat file, in the order in which they are listed in that file.

4.28. PRINT

Format: **print** *file name*
 print pagelen *n*

Abbreviation: none

Operation

The first macro saves to a file the signals and buses listed in the WATCH macro. The signals are saved in the same in which they were listed in the WATCH macro. The page header is printed, together with the signal names. The SETUP Radix macro determines the radix (hex, octal, binary, etc.) of the signals and buses.

The *n* parameter in the second command (**print pagelen** *n*) list the number of lines per page and thus the page length. The saved data can then be printed using the ACTIVE-CAD print option. There is no need to print signals to the screen because they are automatically displayed by the simulator.

Example:

```
print results.dat
print pagelen 24
```

The results.dat file is printed on a page 24 rows lines long.

4.29. QUIT

Format: **quit**

Abbreviation: none

Operation

This command ends execution of the current macro.

Example:

```
quit
```

Terminates execution of the macro and returns control to the simulator program.

4.30. SETUP Radix

Format: **radix** *base vector1 vector2 ... vectorN*

Abbreviation: none

Operation

This macro establishes the radix (hex, octal, binary, decimal) for each signal and bus. This base is used both by the simulators Waveform Viewer and the print operation. The following base abbreviations are allowed:

bin	- binary representation
oct	- octal representation
dec	- decimal representation
hex	- hexadecimal representation

Example:

```
radix oct Data_In
radix hex Address_Bus
```

The Data_In bus values are provided in the octal format and Address_Bus is in the hexadecimal format.

4.31. RELEASE

Format: **release** *signal_or_bus_name*

Abbreviation: **r**

Operation

Disconnects the macros induced signals from the specified signal or bus. Thereafter, the signal or bus will be under direct simulator control.

Examples:

```
release Data_In  
r Address_Bus
```

This causes Data_In and Address_bus to be directly controlled from the simulator.

4.32. SETUP Report

Not implemented

4.33. RESTART

Format: **restart**

Abbreviation: none

Operation

Executes the Power-on operation

Example:

```
restart
```

Causes the simulator Power-on operation. The Power-on setup must be performed from the simulator menus prior to using this macro. This setup may include reset to All Low, All High, Random, Unknown, etc.

4.34. RESTORE

See the *LOAD* macro.

4.35. RUN

Formats: **run n**

run

Abbreviation: none

Operation

Starts the simulation. The simulator performs n clock cycles, and executes signal assignments as defined by the WAVEFORM and PATTERN macro. If clocks are not defined, n number of short STEPs are executed. If the n parameter is omitted, the simulation continues till the longest signal pattern or waveform is processed.

4.36. SAVE

See the *Save/Load Simulation Results* section in this Chapter.

4.37. SIM

Format: **sim time**

sim

Abbreviation: none

Operation

Starts simulation. The simulation time is specified by the *time* parameter. If the time parameter is not listed, a single short STEP is simulated.

Examples:

```
sim 53ns
sim
sim 7.34us
```

These commands simulate the design for the requested time durations: 53 ns, short STEP, 7.34 us, respectively.

4.38. SIMULATE

Not implemented

4.39. STATISTICS

Not implemented

4.40. STEPSIZE

Format: **stepsize** *time*

Abbreviation: **step**

Operation

Sets the short STEP duration which controls all macro operations, including clock, simulation time, waveforms, etc.

Examples:

```
stepsize 15ns  
step 260ps
```

The basic simulation units (short STEPS) as defined by these macro, are 15 ns and 260 ps, respectively.

4.41. TICKSIZE

Not implemented

4.42. TIME_MEASUREMENT

Not implemented

4.43. SETUP Trace

Not implemented

4.44. TYPE

Not implemented

4.45. SETUP Vector

Format: **vector** *bus_name bus_element_1 ... bus_element_n*

Abbreviation: **v**

Operation

This macro is used to describe a bus. Instead of writing the full bus member names, you can write them in shorthand. For example, instead of entering bus A as a0 a1 a2 a3 a4 a5 a6 a7, you can enter **v A a[0:7]**

Both signal names and pin names can be part of a bus. The created buses are automatically listed in the Signal field of the simulator display. Please note that a signal cannot be a member of more than one bus.

Examples:

```
vector Data_In D0 D1 D2 D3
v ADDRESS_BUS AD[0 : 16]
```

4.46. SETUP Watch

Format: **watch** *signal_name_1 signal_name_2 ... signal_name_n*

Abbreviation: **w**

Operation

Selects which signals and buses should be displayed in the simulators waveform viewer. These signals are also automatically selected for printing.

Examples:

```
watch Data_Input
w rst clr enable
```

The first macro has selected **Data_Input** to the display list. The second macro has selected signals **rst**, **clr** and **enable**.

4.47. SETUP Viewwave

Not implemented

4.48. WAVEXFER

Not implemented

4.49. SETUP Wfm Aperiodic

Formats:

```

wfm signal_or_bus_name at_time_1 at_time_2 ... at_time_n
wfm signal_or_bus_name (at_time_1 at_time_2 ... at_time_n)*m
wfm signal_or_bus_name file name

```

Abbreviation: **w**

Operation

In all formats, times are expressed in 100 ps units, unless explicitly stated otherwise.

The first format defines the signal states that are forced upon the signal line at specified time instances. The parameter **at_time_k** can represent time either since the simulation beginning (absolute time) or since the last event. If the parameter **at_time_k** is entered as **@k=1**, then it specifies the absolute time (marker @). If the marker @ is omitted, then the time is specified from the last event. For example, **k=1** specifies High signal state at time **k*100ps** since the last event on that signal line.

If any time is specified as relative, then all following times must also be relative. For example, in *DATA @120=1 @145=0 160=1 185=Z*, the last statement (*185=Z*) must be relative because the preceding one (*160=1*) is relative.

The second format specifies repetitive signal value assignment. All time values in parentheses must be relative. For example, *DATA @0=0 (1000=1 1500=0 2000=Z 1000=0)*5* specifies a signal that is repeated five times, with the values in the parentheses being relative to the preceding signal event.

The third format allows for automatic assignment of signal values (at selected times) from a file.

All macro create timing waveforms with weak signals which control all input pins. When applied at device outputs or nodes, these weak signals are overriddden by strong signals.

Examples:

```

wfm D0 @0=0 @1000=1 @2000=Z @3000=0      (format 1)
wfm A5 @0=H (1000 = L 2000=H) * 10        (format 2)
wfm Data0 events.dat                      (format 3)

```

4.50. SETUP Wfm Decrement

Format: **wfm signal_or_bus_name starting_time starting_value (period= dec by value) * n**

Abbreviation: none

Operation

This macro generates a bus with decrementing values. For example,

*WFM DATA @0=FF\H (10000=dec by 2)*8*

will generate a DATA bus signal that is decrementing each 100 ns (10000x.01ns simulation precision) from its starting hexadecimal value of FF. Since each decrement is by 2, the resulting bus value at the end of the eighth decrement will be EF. If you explicitly list *ns*, e.g. *@100 ns*, then the time is calculated in ns.

Example:

wfm Data_OUT @20ns = 234\D (100ns=dec by 5) * 32

The macro-generated signals are weak signals, and they will be overridden by strong signals present in the node or on output pins.

4.51. SETUP Wfm Divide

Format: **wfm signal_or_bus_name @time1=state1 (period= div by value) * n**

Abbreviation: none

Operation

This macro generates n number of signal values in which each new value equals the current value divided by the *div by* value. Remember that macro operate with natural numbers and e.g. 3-divide by-5 = 0.

Example:

wfm Data_OUT @20ns = 625\D (100ns=div by 5) * 3

This macro generates a signal that starts at 20ns (@=from the origin) with a decimal bus value of 625 (625\D). Each 100ns the current value is divided by 5 to yield a new current value. As a result, the macro produces a waveform:

DATA_OUT @20ns=625 @120ns=125 (absolute time!) 100ns=25 (relative time!) 100ns=5.

This macro is particularly useful in testing binary and other counters.

4.52. SETUP Wfm Increment

Format: **wfm** *signal_or_bus_name* @time1=state1 (period= **inc by value**) * **n**

Abbreviation: none

Operation

This macro generates *n* number of signal values in which each new value equals the sum of the current value and the increment (**inc by value**). The initial or starting time can be expressed as an absolute or relative value.

Example:

```
wfm Data_Bck @20ns = 625\D (100ns= inc by 5) * 3
```

This macro generates at the absolute time of 20ns a bus signal with the initial value of 625 (decimal). Each 100 ns thereafter, the bus value is incremented by decimal 5. As a result, the macro generates a bus:

```
Data_Bck @20=625\D @120ns=630\D @220ns=635\D
@320ns=640\D
```

4.53. SETUP Wfm Multiply

Format: **wfm** *signal_or_bus_name* @time1=state1 (period= **mult by value**) * **n**

Abbreviation: none

Operation

This macro generates *n* number of signal values. Each new value is a product of the current signal value and the multiplier (**mult by value**). The initial or starting time can be expressed as an absolute or relative value.

Example:

```
wfm Data_Bck @20ns = 25\D (100ns= mult by 5) * 3
```

This macro generates at the absolute time of 20ns a bus signal with the initial value of 25 (decimal). Each 100 ns thereafter, the new bus value is equal to the current bus value and the multiplier. As a result, the above macro generates a bus:

```
Data_Bck @20ns=25/D @120ns=125/D @220ns=625/D
@320ns=3125/D
```

4.54. SETUP Wfm Periodic

Format: **wfm** *signal_or_bus_name* @*time1*=*state1* (*time2*=*state2*
time3=*state3*) * **n**

Abbreviation: none

Operation

This macro generates at *time1* the initial signal *state1*. Following this, the macro generates *n* number of signal values described by the expression in parentheses. The initial or starting time can be expressed as an absolute or relative value.

Example:

```
wfm Data1 @20ns = 0 (100ns= 1 120ns=0) * 3
```

This macro generates at the absolute time of 20ns a Low signal, followed by High at @120, Low at @240, High at 340, Low at 460, High at 560 and Low at 680.

4.55. SETUP Wfm Rotate Left

Format: **wfm** *signal_or_bus_name* @*time1*=*state1* (period=**rl by value**) * **n**

Abbreviation: none

Operation

This macro rotates the binary bus value, starting with the *state1* value at the initial *time1*. The bus value rotates to the left at the specified time (period) intervals. The bus shifts by the number of binary positions specified by the (**rl by**) *value*, and there are *n* such shifts altogether. The highest order bit shifts into the lowest order bit, thus performing a closed loop binary-shift operation.

Example:

```
wfm Data1 @20ns = B (100ns=rl by 1) * 3
```

This macro generates at the absolute time of 20ns a bus signal with a value of B (1011\B), followed by 7 (0111\B) at @120ns, E (1110\B) at @220 ns and D (1101\B) at @320ns.

4.56. SETUP Wfm Rotate Right

Format: **wfm** *signal_or_bus_name* @*time1*=*state1* (period=**rr** by *value*) * **n**

Abbreviation: none

Operation

This macro rotates the binary bus value, starting with the *state1* value at the initial *time1*. The bus value rotates to the left at the specified time (period) intervals. The bus shifts by the number of binary positions specified by the (**rl** by) *value*, and there are *n* such shifts altogether. The lowest order bit shifts into the highest order bit, thus performing a closed loop binary-shift operation.

Example:

```
wfm Data1 @20ns = B (100ns=rr by 1) * 3
```

This macro generates at the absolute time of 20ns a bus signal with a value of B (1011\B), followed by D (1101\B) at @120ns, E (1110\B) at @220ns and 7 (0111\B) at @320ns.

4.57. SETUP Wfm Shift Left

Format: **wfm** *signal_or_bus_name* @*time1*=*state1* (period=**sl** by *value*) * **n**

Abbreviation: none

Operation

This macro shift the binary bus value, starting with the *state1* value at the initial *time1*. The bus value shifts to the left at the specified time (period) intervals. The shifting is by the number of binary positions specified by the (**sl** by) *value*, and there are *n* such shifts altogether. The highest order bit is shifted out and lost.

Example:

```
wfm Data1 @20ns = B (100ns=sl by 1) * 3
```

This macro generates at the absolute time of 20ns a bus signal with a value of B (1011\B), followed by 6 (0110\B) at @120ns, C (1100\B) at @220ns and 8 (1000\B) at @320ns.

4.58. SETUP Wfm Shift Right

Format: **wfm** *signal_or_bus_name* @*time1*=*state1* (period=**sr by value**) * **n**

Abbreviation: none

Operation

This macro shifts the binary bus value, starting with the *state1* value at the initial *time1*. The bus value rotates to the left at the specified time (period) intervals. The shifting is by the number of binary positions specified by the (**sr by value**), and there are *n* such shifts altogether. The lowest order bit is shifted out and lost.

Example:

```
wfm Data1 @20ns = B (100ns=sr by 1) * 3
```

This macro generates at the absolute time of 20ns a bus signal with a value of B (1011\B), followed by 5 (0101\B) at @120ns, 2 (0010\B) at @220ns and 1 (0001\B) at @320ns.

4.59. FORCE Unknown

Format: **x** *signal_or_bus_name*

Abbreviation: none

Operation

This macro assigns Unkn_X to a signal or bus.

Example:

```
x U21.INPUT
```

This macro assigns the X (unknown) signal state to the U21.INPUT pin.

5. SPECIAL ALDEC MACRO OPERATIONS

These macro operations are available only with ACTIVE-CAD software. They give you additional test vector editing power and make your work easier and more productive.

5.1. CHECK Design

Format: **chk_design** *design_name*

Abbreviation: **chkds**

Operation

This macro checks if the currently simulated design state is identical to the one specified in the *design_name*. If the design state differs from the one listed in the *design_name*, then the macro is stopped and the message Check Design failed is displayed. If there is no difference, then the next macro from the macro file is performed.

Note: This macro checks only the design timing. It does not compare the hex files, JEDECs and other design elements that have been used in the current and reference simulations.

Example:

```
chk_design Design1
chk_design c:\project\proj_ab\prdes
```

5.2. CHECK MemBlock

Format: **chk_memblock** *hex_file_name* *memory_block_name*

Abbreviation: **chkmb**

Operation

This macro checks if the currently simulated data, residing in the selected *memory_block_name* is the same as in the reference *hex_file_name*. If there is any difference, the macro displays the message Check Mem Block Failed and stops the execution of other macro from the macro file. If there is no difference, then the next macro from the macro file is performed.

Example:

```
chk_memblock Blok1 Main_RAM
chk_memblock c:\project\proj_ab\blkcont  scratch_mem
```

5.3. CHECK MemChip

Format: **chk_memchip** *hex_file_name* *memory_device_name*

Abbreviation: **chkmc**

Operation

This macro checks if the currently simulated data, residing in the selected *memory_device_name* is the same as the data in the reference *hex_file_name*. If there is any difference, the macro displays the message Check Mem Chip Failed and stops the execution of other macro from the macro file. If there is no difference, then the next macro from the macro file is performed.

Example:

```
chk_memchip RAM1 U34
chk_memchip c:\project\proj_ab\mem1 U32
```

5.4. CHECK Timing

Format: **chk_timing** *timing_file_name*

Abbreviation: **chktm**

Operation

This macro checks if the currently simulated data is the same as the data in the reference *timing_file_name*. If there is any difference, the macro displays the message Check Timing Failed and stops the execution of other macro from the macro file. If there is no difference, then the next macro from the macro file is performed.

Since the simulation results depend upon such parameters as on the simulation step, simulation mode, etc., the simulation results are matching the reference only if these parameters are identical to the ones used for generating of the reference *timing_file_name*.

Example:

```
chk_timing restart
chktm c:\project\proj_ab\memload
```

5.5. GET Netlist

Format: **get_netlist** format netlist_file

Abbreviation: **getnet**

Operation

This macro loads a design netlist in the specified *format*. The netlist directory, name and extension are specified by *netlist_file*. If the directory is not listed explicitly, it is assumed that the netlist resides within the current project. The accepted netlist formats are listed below:

<input type="checkbox"/> susie_5	Susie 5.0
<input type="checkbox"/> app_bravo	Application Bravo
<input type="checkbox"/> cadnetix	Cadnetix
<input type="checkbox"/> capfast	CapFast
<input type="checkbox"/> case_tech	Case Technology
<input type="checkbox"/> des_comp	Design Computation
<input type="checkbox"/> futurenet	Futurenet
<input type="checkbox"/> omation	Ovation
<input type="checkbox"/> orcad	OrCad
<input type="checkbox"/> pcad_p	P-CAD Pinlist
<input type="checkbox"/> pcad_h	P-CADH
<input type="checkbox"/> pdif	Pdif
<input type="checkbox"/> racal_redac	Racal-Redac
<input type="checkbox"/> tango	Tango
<input type="checkbox"/> visionics	Visionics
<input type="checkbox"/> viewlogic	ViewLogic
<input type="checkbox"/> wintek	Wintek I
<input type="checkbox"/> susie_ascii	Susie 6.0 Ascii format
<input type="checkbox"/> xilinx	Xilinx
<input type="checkbox"/> edif	Edif 2 0 0
<input type="checkbox"/> susie_bin	Susie 7.0 Binary format
<input type="checkbox"/> lattice	Lattice
<input type="checkbox"/> massteck	Massteck ascii

<input type="checkbox"/> pads	PADS
<input type="checkbox"/> intusoft	Intusoft
<input type="checkbox"/> spice	Spice
<input type="checkbox"/> actel	Actel
<input type="checkbox"/> pcad_n	PCAD Netlist
<input type="checkbox"/> calay	Calay
<input type="checkbox"/> edif_altera	EDIF_ALTERA - MAX+PLUS Edif 2 0 0
<input type="checkbox"/> quick_logic	Quick Logic

Example:

```
get_netlist actel newnet
```

Loads the newnet netlist which resides in the current project directory and is in the Actel format.

5.6. LOAD ASCII

Format: **load_ascii** *timing_file_name*

Abbreviation: **lat**

Operation

This macro loads the ASCII (text) format-based timing signals from the *timing_file_name* into the simulator. The following macro can load additional timings, start simulation, or start simulation with comparison, etc.

Examples:

```
load_ascii atim_1  
lat ascii_tim2
```

5.7. LOAD Design

Format: **load_design** *design_file_name*

Abbreviation: **lds**

Operation

This macro loads the design represented by the *design_file_name* into the simulator. The following macro can load timing files, start simulation, or start simulation with comparison, etc.

Examples:

```
load_design design_1
lds init
```

5.8. LOAD Fault

Format: **load_fault** *timing_fault_file_name*

Abbreviation: **lft**

Operation

This macro loads timing data from the *timing_fault_file_name* into the fault simulator, which is a ACTIVE-CAD option. The following macro can load other timing files, start simulation, or start simulation with comparison, etc.

Examples:

```
load_fault fault_1
lft init
```

5.9. LOAD Fuse Map

Format: **load_fuse** *JEDEC_file_name device_name*

Abbreviation: **lfm**

Operation

This macro loads a JEDEC from the *JEDEC_file_name* into the PLD specified by the *device_name*. Without this JEDEC file, the PLD would simulate as a device with all fuses intact. The following macro can load additional PLDs with JEDECs, timing files, start simulation, or start simulation with comparison, etc.

Examples:

```
load_fuse decoder U23
lfm init PAL1
```

5.10. LOAD Memory Block

Format: **load_memb** *block_hex_file_name* *memory_block_name*

Abbreviation: **lmb**

Operation

This macro loads the hex code from the *block_hex_file_name* into the memory devices specified by the *memory_block_name*.

Examples:

```
load_memb  ram_cnts ram
lmb romst  rom
```

5.11. LOAD Memory Chip

Format: **load_memch** *device_hex_file_name* *memory_device_name*

Abbreviation: **lmc**

Operation

This macro loads the hex code from the *device_hex_file_name* into the memory device specified by the *memory_device_name*.

Examples:

```
load_memch  ram_cnts M5
lmc romst  M12
```

5.12. LOAD Selective Preset

Format: **load_preset** *preset_file_name*

Abbreviation: **lsp**

Operation

The simulator loads the design preset conditions from the file specified by *preset_file_name*. The STARTUP macro executes the actual preset of the design.

Examples:

```
load_preset  initpres
```



```
lsp clear
```

5.13. LOAD Timing

Format: **load_timing** *test_vector_file_name*

Abbreviation: **ltm**

Operation

The simulator loads the simulation test vectors from the file specified by the *test_vector_file_name*.

Examples:

```
load_timing  init  
ltm clear
```

5.14. SAVE ASCII

Format: **save_ascii** *ASCII_file_name*

Abbreviation: **sat**

Operation

The simulator saves the simulation timings (test vectors) as a text (ASCII) in the *ASCII_file_name*.

Examples:

```
save_ascii  new_tim  
sat init_tim
```

5.15. SAVE Design

Format: **save_design** *design_file_name*

Abbreviation: **sds**

Operation

The simulator saves the design simulation conditions and results in the *design_file_name*. For example, the simulator saves the operational mode, simulation steps, JEDECs, hex files, initialization conditions, externally loaded netlists, timings, etc. The saved design data allow for a complete and flawless continuation of the design simulation.

Examples:

```
save_design des_1
sds ddd
```

5.16. SAVE Fault Timing

Format: **save_fault** *design_file_name*

Abbreviation: **sft**

Operation

Simulator saves in the *design_file_name*, the current simulation timings (test vectors) for future fault simulations.

Examples:

```
save_fault error
sft test
```

5.17. SAVE Memory Block

Format: **save_memb** *block_hex_file_name* *memory_block_name*

Abbreviation: **smb**

Operation

This macro saves the contents of the memory block, specified by *memory_block_name*, into a file specified by the *block_hex_file_name*.

Examples:

```
save_memb initial ram
smb simul blok1
```

In the first example, the contents of the memory block *ram* is saved in the *initial* hex file. In the second example, the contents of the memory block (*blok1*) is saved in the *simul* hex file.

5.18. SAVE Memory Chip

Format: **save_memch** *device_hex_file_name* *memory_device_name*

Abbreviation: **smc**

Operation

This macro saves the contents of the memory device, specified by *memory_device_name*, into a file specified by the *device_hex_file_name*.

Examples:

```
save_memch s220 M23  
smc init M15
```

In the first example, the contents of the memory device M23 is saved in the s220 hex file. In the second example, the contents of the memory device M15 is saved in the init hex file.

5.19. SAVE Selective Preset

Format: **save_preset** *preset_file_name*

Abbreviation: **ssp**

Operation

This macro saves the current preset conditions, entered manually under the simulators **Options** menu, in a file specified by the *preset_file_name*.

Examples:

```
save_preset mypreset  
ssp dr21
```

In the first example, the currently used design presets are saved in the mypreset file. In the second example, the design presets are saved in the dr21 file.

5.20. SAVE Timing

Format: **save_timing** *timing_file_name*

Abbreviation: **stm**

Operation

Simulator saves the current simulation timings (test vectors) in the *timing_file_name*. *The timings are stored in the binary format.*

Examples:

```
save_timing aftclk
```

stm aft25ns

Chapter 6

Breakpoint Operations

A breakpoint is a software routine that checks for selected signal conditions in the design. When these conditions are met, you have reached a breakpoint condition and may perform the following design operations:

- ☐ Stop simulation.
- ☐ Place a marker on the screen.
- ☐ Place a milestone (save design status).
- ☐ Save test vectors.
- ☐ Load a new test vector file.
- ☐ Modify the existing test vectors (**Append** operation).

Breakpoints are very handy in tracking major design activities and error conditions. They are used extensively both in hardware tools such as logic analyzers and hardware emulators, and in software development systems.

To create breakpoints, select the **Breakpoints** option from the **Utilities** menu. ACTIVE-CAD responds by displaying the **Breakpoint Options** menu (Figure 6-1). These options perform the following functions:

- ☐ **Breakpoints ON** activates the previously set breakpoint conditions.
- ☐ **Breakpoints OFF** deactivates breakpoint detection.
- ☐ **Edit** invokes the breakpoint editor.
- ☐ **Load** loads previously edited test vector files.

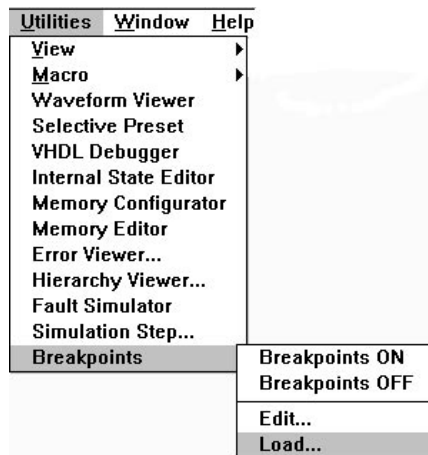


Figure 6-1. Breakpoint Options Menu

Breakpoint editor

Clicking on the **Edit** option in the **Breakpoint Options** menu (Figure 6-1) invokes the **Breakpoint Conditions** window (Figure 6-2), whose **Signals** field shows all signals currently displayed in the simulator.

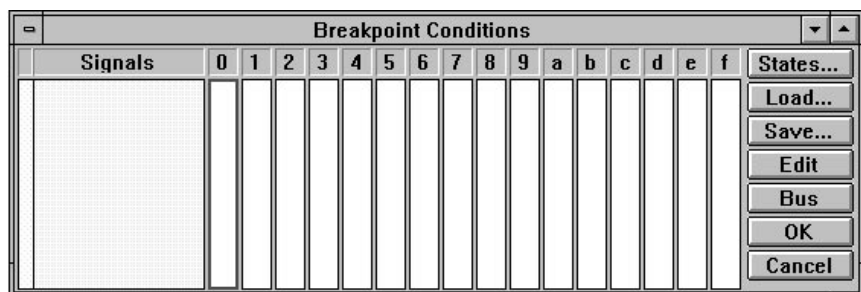


Figure 6-2. Breakpoint Conditions Window

Options

The **Breakpoint Conditions** window contains the following action buttons:

- ☐ **States** allows you to specify the signal logical states for which you want to search.
- ☐ **Load** loads the previously saved breakpoints file.

- ☐ **Save** saves current breakpoints.
- ☐ **Edit** allows you to create the breakpoint program.
- ☐ **Bus** toggles buses between discrete and hex display.
- ☐ **OK** activates the breakpoint program(s).
- ☐ **Cancel** cancels all breakpoint setups and operations.

Breakpoint programs consist of two (2) steps:

1. Creating breakpoint conditions for the selected signals.
2. Building breakpoint programs based on the breakpoint conditions.

Creating a Breakpoint Condition on an Individual Signal

1. Select the first signal in the **Signals** field.
2. Click on a **Conditions** column, e.g. 0. A red frame now highlights this column.
3. Click on the **States...** button. ACTIVE-CAD displays the **Breakpoint States** window (Figure 6-3), which you will use to assign logical states to the signals you have selected. The signals will then be used to create breakpoint programs.
4. In the **Breakpoint States** window click on the button representing the signal logical level you want. To select a signal transition instead of a signal level, click first on the button representing the signal logical level before the transition, and then on the button representing the signal logical level after the transition.
5. Select other signals from the **Signals** field and perform steps 2 through 4 for each one.
6. When completed, the **Breakpoint Condition** will represent the logical AND function of all selected signals.

You can define up to fifteen (15) **Breakpoint Conditions** (0 through F) and use them in your breakpoint program.

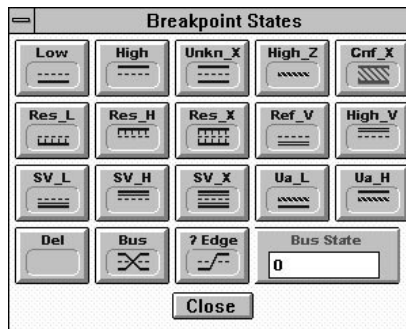


Figure 6-3. Breakpoint States Window

Creating Breakpoint Conditions on Buses

You can assign a breakpoint condition to individual bus signal lines and to an entire bus line (global bus operations).

Individual bus line breakpoint conditions

To set breakpoint conditions on individual signal lines, click on the **Bus** button. It automatically displays buses as sets of signal lines. You can set breakpoint conditions on these lines as described in **Creating a Breakpoint Condition on an Individual Signal**, above.

Global bus line breakpoint conditions

The bus must be shown as a hex bus signal in the **Signals** field. If the signal line is split into individual lines, toggle the **Bus** button. Next, follow these steps:

1. Click on the selected bus signal line in the **Signals** field.
2. Click on the selected **Conditions** field.
3. Click on the **States** button. When the **Breakpoint States** window appears, enter the bus hexadecimal value into the **Bus State** field and then click on the **Bus** button. This button operates as a confirmation of the bus setup. If you want to set a breakpoint on a condition when all bus signal lines transition, click on the **?Edge** button.
4. If you want the simulator to stop when it detects the bus breakpoint condition, click on the **OK** button. If you want to create a more complex breakpoint program, click on the **Edit** button to invoke the **Breakpoint Edit** window (Figure 6-4). The **Programming Process** section below describes how to use this window. Click on the **Cancel** button to delete the breakpoint setups.

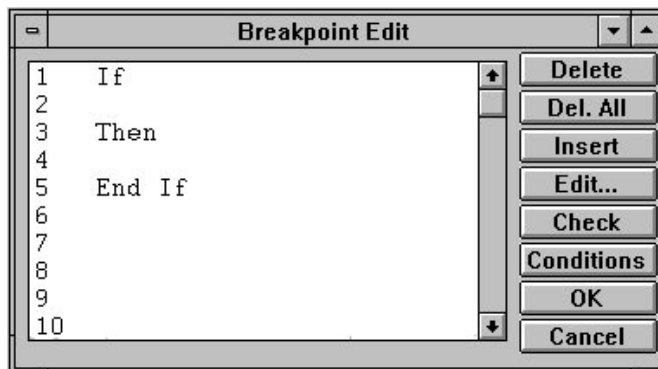


Figure 6-4. Breakpoint Edit Window

Breakpoint Edit Window Options

- ☐ **Delete** deletes the selected instruction, argument or empty line.
- ☐ **Delete All** deletes the entire breakpoint program.
- ☐ **Insert** inserts an empty line.
- ☐ **Edit** displays a window with programming options.
- ☐ **Check** verifies that the program is correct.
- ☐ **Conditions** returns you back to the **Breakpoint Conditions** window.
- ☐ **OK** activates the breakpoint detection process.
- ☐ **Cancel** cancels the program in process, but does not delete it.

Programming Process

All operations described below refer to the **Breakpoint Edit** window:

1. When the **Breakpoint Edit** window appears, it generally lists the *If ...Then...If* construct. If it is not listed, position the cursor on the first line and double-click. An **Instruction List** window appears which lists two basic instructions (*If* and *If Else*). Double-clicking on one of these instructions transfers the instruction into the **Breakpoint Edit** window and leaves empty rows in which you may enter the instruction arguments (e.g. *IfThenEnd If*).
2. To nest instructions, double-click on the *If* or *End If* instructions, producing the **Instruction List** or the **Instruction-Argument List** window, respectively. By double-clicking on the instruction you need, or by highlighting the instruction and then clicking on the **Set** and **Close** buttons, you will add this instruction to the breakpoint program.
3. When you double-click on the line located below the *If* (or *Else If*) instruction, ACTIVE-CAD displays the **Conditions List** window. To

simplify programming, the window lists only two options: *Condition* and *Not Condition*. Clicking on the **Expand** box will produce additional instructions in this window (Figure 6-5).

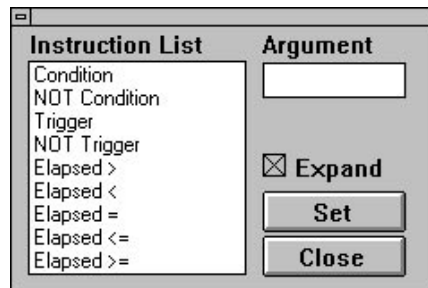


Figure 6-5. Expanded Conditions List Window

- ☐ **Condition** activates the program if the *Condition* occurred.
 - ☐ **NOT Condition** activates the program if the *Not Condition* occurred.
 - ☐ **Trigger** activates the program if the *Trigger* occurred.
 - ☐ **Not Trigger** activates the program if the *Not Trigger* occurred.
 - ☐ **Elapsed** activates the program after an indicated time has lapsed.
4. If you double-click on the *Then* instruction, the **AND/OR Instruction** window will appear and allow you to specify AND and OR logical operations on several conditions defined in the **Breakpoint Conditions** window. Do not use this option if you are using only one condition in the breakpoint program.
 5. Double-clicking on the line located below the *Then* instruction invokes the **Instructions-Actions List** window, which lists actions available to ACTIVE-CAD upon breakpoint detection. Initially this window lists only the actions most frequently used. However, if you click on the **Expand** box, you will see additional options (Figure 6-6). You need to select the action you want, e.g. **Mark Breakpoint**, **Save Test Vector**. For example, if you select **Mark Breakpoint** and enter the letter *q* into the **Argument** field, then each time that ACTIVE-CAD detects a breakpoint, it will mark the breakpoint with the letter *q*. You can select several actions to be executed one after another at a breakpoint condition.

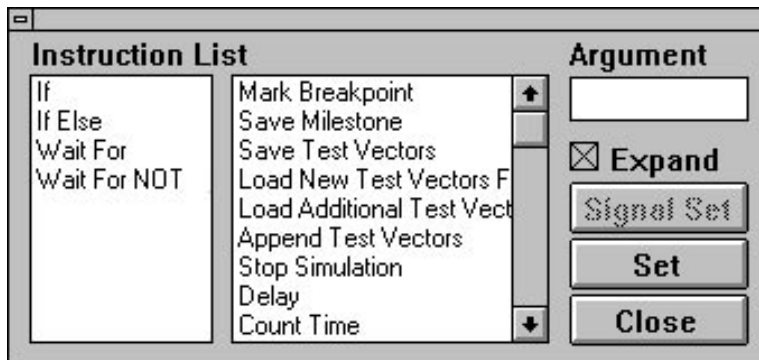


Figure 6-6. Expanded Instructions-Actions List Window

- ☐ **Mark Breakpoint** marks the location of the breakpoint occurrence.
 - ☐ **Save Milestone** saves the design status and marks the display.
 - ☐ **Save Test Vectors** saves existing signal wave forms as test vectors.
 - ☐ **Load New Test Vectors File** loads a test vector file.
 - ☐ **Load Additional Test Vectors** adds additional signals and test vectors to the screen.
 - ☐ **Append Test Vectors** loads test vectors starting at the blue cursor.
 - ☐ **Stop Simulation** stops the simulation when it reaches a breakpoint.
 - ☐ **Delay** delays the action following the command by a specified time.
 - ☐ **Count Time** starts counting time.
 - ☐ **Set Trigger** marks certain signal conditions or program branches.
 - ☐ **Clear Trigger** clears the triggers you have set.
6. Click on the **Set** button to complete the selection of an instruction, condition or action. Click on the **Close** button to complete or terminate the breakpoint programming process.
 7. Click on the **OK** button in the **Breakpoint Edit** window. ACTIVE-CAD will check the breakpoint program and report any errors so that you can correct them right away.
 8. Click on the **OK** button in the **Breakpoint Conditions** window. This sets the **Breakpoint ON** condition in the **Utilities/Breakpoint Menu** (Figure 6-1).

Appendix A

ASCII Timing Format Specification

ALDEC ASCII TIMING is a text file used to save and load timing for selected signals and/or pins.

ASCII Timing File structure

ALDEC ASCII TIMING is composed of three blocks, each followed by a block designator. Block designators are made of the pound sign (#) and the name, followed by CR/LF. The sequence of data blocks is fixed and listed below:

```
#Hier  
#Signals  
#Timing  
#End
```

Any line beginning with a semicolon (;) is a comment and will be ignored by the timing loader. Comments are allowed within any line and in any part of the file.

#Hier block

This block lists all hierarchical levels used in the Timing diagram by means of identifiers. An identifier is made of a @ followed by a number. You should assign the identifiers to hierarchical blocks. The assignment sequence is not important because it is used for housekeeping purpose only. However, all identifiers should be listed in the **#Hier** block in ascending order, starting with @1 for the design ROOT.

The hierarchy name begins with **ROOT** followed by names of the following levels, with spaces between each level.

Example:

```
#Hier
@1 ROOT          ; ROOT has identifier @1
@3 ROOT A001 A1   ; this block has been given identifier @3
```

The **#Hier** block lists only the hierarchies used in the test vector definition. The above example has two hierarchy identifiers: @1 and @3. **ROOT** is the name of the top hierarchical level of the design, **A001** is the name of the next lower hierarchical level (child of the ROOT), **A1** is the name of the next lower hierarchical level (child of A001).

#Signals block

This block lists all pins and signals included in the Timing diagram. They are listed in the same order here as on the ACTIVE-CAD screen (from top to bottom). The pin/signal description includes a number followed by the hierarchy identifier (@1, etc.) and the pin/signal name. The pin name (e.g. U23 X1) is composed of the component name and pin name, separated by a space. The signal name (e.g. OUT1) is a one-word identifier. Pins/signals should be listed in the display (ascending) order, starting with 1. The example below shows three signals in their display order:

```
#Signals
1 @3 U23 X1      ; pin X1 of U23, located at @3 hierarchical level
2 @1 OUT1        ; signal OUT1 at the @1 hierarchical level (lowest)
3 @3 U2 14       ; pin 14 of U2, located at @3 hierarchical level
```

The example defines three signals, wherein:

1, 2, 3 are sequential signal numbers,

@1, @2, @3 are hierarchy identifiers that show each signal location in the design hierarchy

U23, U2 are part names, from the specified hierarchy levels

X1, 14 are pin names of the specified part, and they define the signal,

OUT1 is the name of the signal from of the specified hierarchical level and is the equivalent to the signal name in ACTIVE-CAD.

NOTE: Any signal defined by part and pin name is comprised of four elements (e.g. 1 @3 U23 X1), and any signal defined only by its name is comprised of three elements (e.g. 2 @1 OUT1).

#Timing block

This block lists timing for pre-specified pins and signals. Each line specifies the time and state of all signals. Time is the number of nanoseconds from the beginning of timing (0ns). You can use the following formats to specify time:

10	10 nanoseconds
10.2	10.2 nanoseconds
72p	72 picoseconds
2.1u	2.1 microseconds

Use the following time scale designators:

blank	nanoseconds
n	nanoseconds
p	picoseconds
u	microseconds

The logical state is represented by one digit or letter, as shown in the table below:

1	input high
0	input low
H	output high
L	output low
X	unknown
Z	high impedance
R	reference voltage
V	high voltage
.	(period) designates no Timing

There is a space separating the states for each signal. The logical state of the first signal begins in the same column in every line (e.g. 15th column) in order to provide clarity.

Example of format:

```
#Timing
; 1 2 3 this comment line may be provided for ref-
erence
0 1 H X ;each row lists states of all signals
at the specified time
10.2 1 L X
21 0 L X
30 1 H X
40 1 . H
```

```

|   |   |
;   |   3rd column lists the signal #3 test vector
;   |   1st column lists the signal #1 test vector

```

This listing represents specific time events, listed in the left column. They are expressed in nanoseconds (default).

#End

This keyword ends the timing file. Any information following **#End** is ignored.

Example

Following is an example of a Timing diagram and its ASCII Timing conversion.

Timing diagram:

```

U23.X1 _ ; this graphics represents signal waveform of U23.X1pin
OUT1  -__ ; this graphics represents OUT1 signal waveform
U2.14 xxxx-;this graphics represents signal waveform of of U2.14 pin

```

ASCII equivalent of the timing:

#Hier

@1 ROOT

@3 ROOT A001 A1

#Signals

1 @3 U23 X1 ; this signal will be listed in the 1st test vector column

2 @1 OUT1 ; this signal will be listed in the 2nd test vector column

3 @3 U2 14 ; this signal will be listed in the 3rd column, below

NOTE: To better differentiate between inputs and outputs, all input signals have 1s and 0s. Output signals have Hs and Ls, respectively.

#Timing

```

;      1 2 3
0      1 H X
10.2   1 L X
21     0 L X
30     1 H X
40     1 H H

```

#End

NOTE: Both lower and upper case characters can be used interchangeably in the Timing file, with the exception of the pin name.

Appendix B

ALDEC ASCII Netlist Format

The ALDEC Netlist is a text file used as an intermediate format for netlist loading. This netlist includes all information about hierarchy, components, connections, line delays, name aliasing and programmable block configuration.

Netlist Structure

The ALDEC netlist is comprised of several data blocks (some of them optional), each preceded by a block designator. Block designators are built from a pound sign (#) and a 4-character name, and must be followed by a space, TAB or CR/LF. A netlist defines all blocks in the following sequence:

```
#File
#Qty
#Type
#Comp
    #conf
#Sign
...
#Node
...
#EndN
```

Any text enclosed in brackets (< >) will be treated as a comment and will be ignored by the netlist loader. Comments can be placed anywhere. Both DOS and UNIX file formats are legal. All names (types, component names, pin names) are enclosed in quotation marks. Block fields are ter-

minated by semicolons; commas separate field records. Space, TAB and LF (or CR/LF) characters may be placed anywhere because they are ignored by the loader. Names enclosed in quotation marks (") cannot include spaces, TABs or CR/LFs; any other character in the name is legal. All words can be written in upper or lower case letters. However, names within quotation marks (") are case sensitive.

List of special characters

block designator marker
 <> *comment delimiters*
 " " *name delimiters*
 , *record separator*
 ; *field terminator*
 () *line delay delimiters*
 . *separates component number from component pin number or name*

Section #FILE

This block specifies a source netlist file name (i.e. the one from which the netlist was created). The filename **cannot** include an extension.

Format:

#File "file_name";

Example:

#File "TEST1";

Section #QNTY

This block specifies the number of components, nested blocks, signals, connections, and/or line delays in the netlist.

Format:

#Qty no_of_components, no_of_nested_blocks, no_of_signals,
 no_of_types, no_of_delays, no_of_nodes;

Where:

no_of_components - total number of all netlist components
 no_of_nested_blocks - number of nested blocks
 no_of_signals - number of signals (labels and terminals)
 no_of_types - number of type names
 no_of_delays - total number of line delays specified within

no_of_nodes parentheses “()”, in all #Node block fields
 total number of #Node blocks

Examples::

```
#Qty 120,12,21,36,0,126;
#Qty 15,0,9,4,5,155;
#Qty 120, 12, <signals> 36, <types >21, <delays> 5, 155;
```

You must list all six blocks in the Block Section.

Section #TYPE

#Type block specifies the list of component types used in the netlist.

Format:

```
#Type
No, "type_name";
...
No, "type_name";
```

Where:

No is the sequential type_name number (1, 2, ...);

Example:

```
#Type
1, "SN74LS75N"      ; this is a chip type name
2, "AmPAL20R8-10"; comments can be inserted in lines
...
20, "ADDER-4"      ; this is a nested block file name
```

Section #COMP

This block lists all netlist components, except signals and their class (Nested Block or Chip). Programmable components can also be described here within #CONF subblock. All components should be listed in alphabetical order.

Format:

```
: #Comp
No, "name", "location", type_No, class;
...
No, "name", "location", type_No, class;
```

Where:

No - sequential name number (1, 2, ...)
 name - component name (e.g. 7400)

location - component location (e.g. U21)
 type_No - component type name number specified in #Type block
 class: - C = chip or cell, N = nested block.

Examples:

#Comp

```
1, "P101", , 2, C;
2, "US12", <no location>, 1, C;
3, <name> "US15", < no location>, <type> 1, C;
...
```

#Comp

```
1, "IN1", "P11", 9, C <both name and location present>
#Conf ... ;
2, "P15", 9, C <name only>
#Conf ... ; ...
```

Section #CONF

This is the only subblock in the netlist. It is optional and allows you to specify programmable cells and logic block configuration. The format of this subblock is flexible because the configuration descriptions are different for different cells. The first field specifies the configuration code, the second one lists the #conf subblock terminator (character). These two fields are followed by the configuration description lines; comments are not legal within these lines.

The following is a list of configuration codes and what they stand for, as per LCA and XNF specification (all for XILINX FPGA):

```
1 - OSCx - internal oscillator
2 - TBUF - internal tri-state buffer
3 - CLB2 - 2000 family CLB
4 - IOB2 - 2000 family IOB
5 - CLB3 - 3000 family CLB
6 - IOB3 - 3000 family IOB
7 - Xilinx 4000 family ROM/RAM initial contents
```

Format:

```
#Comp No, "name", "location", type_No, class
      #Conf code, "terminator", configuration lines
terminator ;
```

Examples:

```
#Comp 1, "TCO", "AD", 12, C
```

```

#Conf 5, "#",
Base FG
Config X:F Y:G RSTDIR: ENCLK: DX: DY: CLK: F:A:B:C:D G:A:B:C
Equate F=(A*B*C*D)
Equate G=(C*A*B)
#;
2, , "TBUF.AP.1", 38, C #Conf 2, ".", 1 .;
3, , "TBUF.AA.1", 38, C #Conf 2, ".", 0 .;

```

Section #SIGN

This block lists all signals and their types.

Format:

```

#Sign
No, "name", type, bus;
...
No, "name", type, bus;

```

Where:

No - signal number (N+1, N+2, ...), N is the last number in #Comp block;
 name - signal name;
 type - signal type; Table B-1 lists all the signal types
 bus - bus number; Members of the same bus have the same number
 (1 - bus 1, 2 - bus 2, etc.). Discrete signals have no number or
 their number is 0.

Examples:

```

#Sign
58, "INP1", I, 1; <name numbering continues>
59, "INP2", I, 1; <signals INP1 and INP2 are members of same bus>
60, "I/O_A", B,; ...

```

NOTES:

- ☐ A terminal is unidirectional if it is not connected to a chip or cell pin.
An example of such a connection is from terminal to terminal:
TERM_1>-----< TERM_2
If the source netlist does not indicate a terminal type, the terminal will be listed as unidirectional.
- ☐ The #Sign block lists bus LSB (least significant bit) first, MSB is the last.
- ☐ Only bus signals can be listed between the LSB and MSB bus members within the #Sign block.

Table B-1. List of Signal Types.

Signal Type	Description
I	input terminal,
O	output terminal,
B	bi-directional terminal,
U	unidirectional terminal,
L	internal label,
1	HIGH state terminal (e.g. Vcc),
0	LOW state terminal (e.g. GND),
H	high voltage terminal (e.g. Vpp),
R	reference voltage terminal (e.g. Vbb).

Section #NODE

There are as many #Node blocks as nodes in a netlist. Each of them lists components wired together.

Format:

#Node

Comp_No.pin, Comp_No.pin(delay), Comp_No.pin(delay),... ;

Where:

Comp_No component number (as listed in #Comp or #Sign);
 pin pin name or number, 0 for signals;
 (delay) line delay in [ns]; this record is optional.

Examples:

#Node

10."AD0" <name>, 25.4 <number>, 47.0;

#Node

35."CS", 47."CS", 28."Y1";

#Node

10."O", 18."A"(3), 17."DI"(0), 25."B", 37."B"(4.7); <this node has line delays>

Section #ENDN

This keyword ends the netlist file. The ending “;” can be omitted.

Notes

All blocks except #File, #Qty and #EndN are optional. If there is nothing to list, the entire block will be omitted.

Appendix C

Library Listing

The following is an abbreviated list of SUSIE libraries. The parts can be available in all technologies, packaging and from all manufacturers.

TTL	77	164	266	456	621	777	1604
	78	165	269	465	622	779	1645
	83	166	273	466	623	786	1760
00	85	167	276	467	638	804	1760A
01	86	168	279	468	639	805	1762
02	89	169	280	490	640	808	1779
03	90	173	282	518	641	810	1894
04	91	174	283	519	642	811	1895
05	92	175	286	520	643	821	1896
06	93	178	292	521	644	822	1897
07	94	180	294	522	645	823	2232
08	95	181	295	526	646	824	2233
09	96	182	297	527	647	825	3037
10	97	183	298	528	648	826	3038
11	107	190	299	533	649	827	3040
12	109	191	348	534	651	828	3893
14	111	192	350	537	652	832	5074
15	113	193	352	538	653	841	8003
20	116	194	353	539	654	842	8960
21	120	195	354	540	655	843	8961
22	121	196	355	541	656	844	30240
25	122	197	356	543	657	845	30244
26	123	198	365	545	673	846	30245
27	125	199	366	547	674	850	30640
30	126	221	367	548	676	851	50109
31	128	229A	368	560	677	861	50728
32	131	225	373	561	678	862	50729
33	132	230	374	563	679	863	
34	133	231	376	568	680	864	
36	134	232A	377	569	682	866	
37	135	233A	378	574	685	867	
42	136	234	379	575	686	869	
45	137	235	381	577	687	870	
46	138	236	382	579	688	873	
48	139	240	385	583	689	874	DIP314A
49	140	241	386	588	711	876	DIP314B
50	143	242	390	590	712	878	DIP316A
51	147	243	393	591	723	879	DIP316B
53	148	244	396	592	725	881	RES
54	150	245	398	593	732	882	RPAK10
55	151	246	399	594	733	899	RPAK14
56	153	248	410	595	746	990	RPAK16
57	154	250	412	596	747	991	RPAK8
64	155	251	422	597	755	996	SIP106A
65	156	253	423	598	756	1000	SIP106B
68	157	256	432	604	757	1004	SIP108A
69	158	257	436	605	758	1032	SIP108B
70	159	258	440	606	759	1241	SIP110A
72	160	259	441	607	760	1242	SIP110B
73	161	260	442	614	762	1243	
74	162	264	444	615	763	1244	
75	163	265	455	620	776	1245	
76							

List of available libraries:

- ☐ **8051**(Intel 8051, 8052, 8031, 8032)
- ☐ **ACTEL1, ACTEL2, ACTEL3** (Actel FPGA macro library)
- ☐ **ATMEL** (Atmel EPLDs)
- ☐ **ALTERA** (Altera MAX & FLEX)
- ☐ **CMOS** (TTL-CMOS & CD4000)
- ☐ **ECL** (10K, 100K, 10H, 10E)
- ☐ **EPLD** (Altera & Intel EPLD, e.g. EP300, EP1800)
- ☐ **ISA** (Intel ISA Simulation library)
- ☐ **LATTICE** (Lattice pLSI macro library)
- ☐ **M6809** (Motorola MPU)
- ☐ **MACH** (AMD CPLD, MACH1, 2, 3 & 4)
- ☐ **MEMORY** (RAM, ROM, SRAM, DRAM, FIFO, Dual Port, etc.)
- ☐ **PASSIVE** (Resistors, Capacitors, Inductors, Switches)
- ☐ **PERIPHE** (Intel and Motorola Peripheral Devices)
- ☐ **PLD** (Classical PLD and PAL devices e.g. 10H8, 22V10)
- ☐ **QUICKLOG** (Quicklogic PASIC Macro library)
- ☐ **TTL** (LS, S, F, ALS, etc.)
- ☐ **XILINX2, XILINX3, XILINX4** (Xilinx FPGA macro library)
- ☐ **X2000U, X3000U, X4000U, X7000U, XBLOXU** (Xilinx Unified Libraries)
- ☐ **Z80** (Zilog Z80 MPU)
- ☐ **AMD** (symbols only, AMD devices)
- ☐ **ANALOG1** (symbols only, Analog devices, amplifiers)
- ☐ **DEVICE1** (symbols only, discrete components, transistors, diodes)
- ☐ **DIP** (symbols only, footprint symbols)
- ☐ **MEM** (symbols only, Additional memory devices)
- ☐ **MICRO** (symbols only, Microprocessors)
- ☐ **PAL** (symbols only, additional PAL & PLD symbols)
- ☐ **SEMI** (symbols only, CD4XXX symbols)

NOTE: Libraries marked as “symbols only” do not contain simulation models.

Appendix D

Error Messages

Netlist Import Messages

This Appendix explains error messages reported by ACTIVE-CAD during loading the netlist into the simulator, and suggests ways to correct the problems.

WARNING: There is no netlist associated with current project

ACTIVE-CAD cannot find the requested netlist on the hard disk. Either an error has been created during netlist conversion or the requested netlist file has been renamed or deleted.

ACTION: *Check if the requested netlist file resides in the selected directory and has the same file name extension as entered. If the file exists, reload the netlist again.*

Unknown component - xxx, yyy

The specified xxx model of the yyy type has not been found in the project libraries. ACTIVE-CAD creates an empty (dummy) model for the selected symbol.

ACTION: *Check if the desired libraries have been included in the project. If not, add them with the help of the **Project Libraries** option. Ignore if you don't have the model.*

Some IC Models not loaded. They were not found or not available in the keylock

ACTIVE-CAD informs you that some of the models were incorrectly loaded. This may happen when some of the model files have been deleted from the pro-

ject and the model cannot be completely loaded. Such models are replaced by dummy models so that ACTIVE-CAD will be able to run the simulation.

ACTION: *Reload the project libraries and load the netlist again.*

Insufficient memory, loading gate array will be aborted

There is not enough RAM memory to load the entire ASIC design. Loading of the ASIC gate array will be aborted.

ACTION: *Free more RAM memory and load the design again.*

Error while loading model; loading gate array will be aborted

A model-related error has been detected during loading of the ASIC design. Loading of the ASIC will be aborted.

ACTION: *First load the correct ASIC library and then load the design again.*

Improper format netlist for gate array

An improper netlist format has been selected for the ASIC design netlist. For example, you may have selected the Schema format instead of the Xilinx XNF format. ACTIVE-CAD will not load the ASIC design till the correct netlist format is specified.

ACTION: *Select the correct netlist format within the **Load Netlist** option.*

xxx - not loaded. Library not found in the keylock

Model xxx has not been loaded because it is not allowed in the keylock. ACTIVE-CAD creates a dummy model so that the simulation may proceed.

ACTION: *Insert a keylock with the desired library enabled.*

Board - bbb, board id - iii, port - ppp, error code - eee

This is an error message related to the Virtual Hardware I/O board installation (PC-DIO-24 or PC-DIO-96). ACTIVE-CAD displays the I/O board and its port ID numbers and the error ID number, as listed in the I/O board documentation.

ACTION: *Reinstall the VHE I/O board as per ALDEC installation procedure.*

Incorrect pin number nnn for hhh, ccc

While loading a netlist ACTIVE-CAD has found that the *nnn* netlist pin number is different from the *hhh* ASIC (library symbol) pin number. ACTIVE-CAD stops loading the ASIC netlist.

ACTION: *Correct either the ASIC symbol (in the Symbol Editor) or correct the netlist.*

Load Netlist for *hhh* , *ccc*

This is an advisory message only; a netlist of an *hhh* ASIC design and the *ccc* type is being loaded.

ACTION: *No action is required.*

Incorrect hierarchy name - *hhh*

There is an error in the hierarchy created by loading an *nnn* hierarchical alias.

ACTION: *Correct (rename) the hierarchical schematic or macro.*

Not enough memory for a new node

ACTIVE-CAD ran out of memory while loading the current netlist node.

ACTION: *Free some more RAM and load the netlist again.*

LDNODE: Node element number equal to zero

Incorrect ID name or number of signal or component in the netlist. Either the wrong netlist format has been used or there is a disk read error.

ACTION: *Check the netlist format and create a new netlist. Or, read the netlist again.*

LDNODE: Cannot read node element identifier

An error occurred while reading a netlist from disk. The hardware disk may have failed or the netlist file may have been corrupted.

ACTION: *Create a new design netlist and check the hard drive.*

LDNODE: There is no pin name and no pin number

An error occurred in the netlist node; missing a pin ID (it could be a wrong netlist format or software bug). It could be created during the netlist read operation or the file has been corrupted.

ACTION: *Check the netlist format. Regenerate the design netlist. If error persists, contact ALDEC.*

LDNODE: Unknown pin name

A discrepancy between the pin names in the netlist and the symbol library has been detected. It could be the result of ACTIVE-CAD searching the component libraries in improper sequence. For example, two components in two different libraries may have the same name yet they may differ in pin designations. If the libraries are in the improper order, a component from the wrong library may be selected.

ACTION: *Check if the libraries are in the proper search sequence and rearrange them if needed. Also, check the netlist format.*

LDNODE: Unknown pin_numbering and pin number specified

The netlist has some pin numbers which are missing in the symbol (library). Either a wrong or incomplete library has been attached to the project or the libraries are in the wrong sequence. Also, a wrong netlist format may have been used.

ACTION: *Check if the desired library has been loaded and if the libraries are in the desired search sequence. Rearrange them if needed. Also, check the netlist format.*

LDNODE: Unknown pin_number

The netlist has a pin number which is missing in the symbol (library). Either a wrong library has been attached to the project, an incomplete library has been attached (some library files may be missing) or the libraries are in the wrong sequence. Also, the wrong netlist format may have been used.

ACTION: *Check if the desired library has been loaded and if the libraries are in the proper search sequence. Rearrange them if needed. Also, check the netlist format.*

LDNODE: Cannot read empty pin name

If the netlist uses pin numbers, then the space for the pin names must be empty. If ACTIVE-CAD does not see the empty pin name space, then either the netlist file is corrupted or there is a hardware error.

ACTION: *Check if the hardware is operational. Next, generate a new netlist.*

LDNODE: Pin name and pin number specified simultaneously

The netlist contains both the pin name and its number. It should have either one or the other but never both. This error could be caused by an erroneous netlist format or by a corrupted netlist file.

ACTION: *Check the netlist format. Generate a new design netlist format.*

LDNODE: Incorrect chip identifier

An erroneous description of a netlist node. It could be caused by the wrong netlist format, a conversion error or file corruption due to hardware failure.

ACTION: *Check the netlist format. Generate a new design netlist format.*

LDNODE: Cannot read pin number

An erroneous description of a netlist node. It is most probably caused by file corruption due to hardware failure.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

LDNODE: Cannot read pin name

An erroneous description of a netlist node. It is most probably caused by hardware failure.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

DUMMY: Node element number equal to zero

An error has been found while writing a dummy model. Either the wrong chip ID or netlist format has been used. It could also be caused by a conversion error or a corrupted netlist file.

ACTION: *Generate a new design netlist format. Also check the chip ID and symbol libraries.*

DUMMY: Cannot read node element identifier

A node processing error has been found while writing a dummy model. This error is most often caused by hardware failure. Sometimes, netlist file corruption may be the cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

DUMMY: Cannot read pin number

A pin number error has been found while writing a dummy model. This error is most often caused by hardware failure. Netlist file corruption may be yet another cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

DUMMY: Cannot read pin name

A pin name error has been found while writing a dummy model. This error is most often caused by hardware failure. Netlist file corruption may be yet another cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

DUMMY: There is no pin name or pin number

Neither pin name nor number have been found while writing a dummy model. This error is most often caused by a wrong netlist format. Sometimes, hardware failure or netlist file corruption may be the cause.

ACTION: *Check the netlist format, then the hardware (disk). Generate a new design netlist format. If the problem persists, contact ALDEC (converter problem).*

DUMMY: Cannot read an empty pin name

If the netlist uses pin numbers, then the spaces for the pin names should be empty. If ACTIVE-CAD does not see the empty pin name spaces while it generates dummy models, then either the netlist file is corrupted or there is a hardware error.

ACTION: *Check if the hardware is operational. Next, generate a new netlist.*

DUMMY: Pin name and pin number specified simultaneously

The netlist contains both the pin name and its number. It could be caused by an erroneous netlist format or by a corrupted netlist file.

ACTION: *Check the netlist format. Generate a new design netlist format.*

LDNODE: Incorrect signal identifier

The netlist contains an incorrect signal name and number. It could be caused by an erroneous netlist format or corrupted netlist file.

ACTION: *Check the netlist format. Generate a new design netlist format.*

Not enough memory for a dummy model table

There is not enough RAM memory to create dummy model tables.

ACTION: *Free RAM memory and then load the netlist.*

DUMMY: Redundant pin name

ACTIVE-CAD found an erroneous node description while creating dummy models. The netlist contains both the signal name and its number. The error could be caused by an erroneous netlist format or by a corrupted netlist file.

ACTION: *Check the netlist format. Generate a new design netlist format*

No inputs in the node

This is only a warning that there is no input signal in the node. Such a node will float during simulation. The floating node could be driven by an analog-to-digital converter, which may not have a model in the library (see *Missing Models* in the *Loading a Design* section, Chapter 2).

ACTION: *Check the design node for a missing terminal and/or output pin. If needed, override the node with a signal waveform which emulates the desired node behavior.*

No outputs in the node

This is only a warning that there is no output signal in the node.

ACTION: *Check the design node for a missing terminal and/or input pin.*

More than one label in the node

This is only a warning that there is more than one signal node name. ACTIVE-CAD will not know for sure which name to use.

ACTION: *Check the design node for multiple signal names.*

More than one normal (Totem_Pole) output in the node

This is a warning that two totem-pole output pins are connected in the same node. This is generally an improper design and should be corrected.

ACTION: *Check the design node for multiple totem-pole outputs in a single node and remove the illegal ones.*

Power point is connected to the output pin

This is only a warning that an output pin is connected to a logical ground.

ACTION: *Check the design node for a grounded output pin.*

Mixed logic level drivers in the node

A warning that outputs from different technology devices (e.g. TTL and ECL) are connected together.

ACTION: *Check the design node for the illegal output pin connection.*

Mixed power source levels in the node

This is a warning that several different power supplies are connected together in the same node, e.g. Vcc and GND.

ACTION: *Check the design node and remove the unnecessary power supply signals.*

NETGEN: Cannot read component identifier

ACTIVE-CAD cannot read component parameters most often caused by bad hardware. Sometimes, netlist file corruption may be the cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format*

NETGEN: Undefined generics

ACTIVE-CAD cannot read component parameters from the netlist because they do not exist in the associated library. You have either used the wrong library or assigned the wrong library search sequence.

ACTION: *Check the project library listing. Add a new library or rearrange the sequence of existing libraries.*

NETGEN: Cannot read generic name

ACTIVE-CAD cannot read component parameters from the netlist. This error is most often caused by bad hardware. Sometimes the netlist file corruption may be the cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format*

NETGEN: Unknown generic name : ggg

ACTIVE-CAD cannot read component parameters from the netlist because they do not exist in the associated library. You have either used the wrong library or assigned the wrong library search sequence.

ACTION: *Check the project library listing. Add a new library or rearrange the sequence of existing libraries.*

NETGEN: Cannot read generic max. value

ACTIVE-CAD cannot read the component generic max. value from the netlist. This error is most often caused by bad hardware. Sometimes netlist file corruption may be the cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

NETGEN: Cannot read generic avg. value

ACTIVE-CAD cannot read the component generic average value from the netlist. This error is most often caused by bad hardware. Sometimes netlist file corruption may be the cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

NETGEN: Cannot read generic min. value

ACTIVE-CAD cannot read the component minimum value from the netlist. This error is most often caused by bad hardware. Sometimes netlist file corruption may be the cause.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

Cannot read package information

ACTIVE-CAD cannot read the component packaging ID from the netlist. This error is most often caused by bad hardware and netlist corruption. Sometimes there can be a discrepancy between the package information and the library component.

ACTION: *Check the hardware (disk). Generate a new design netlist format. If it does not help, check the component packaging information in the library.*

Cannot read chip attribute

ACTIVE-CAD cannot read component parameters from the netlist. This error is most often caused by bad hardware and netlist file corruption.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

Cannot read pin attribute

ACTIVE-CAD cannot read component pin attributes from the netlist. This error is most often caused by bad hardware and netlist file corruption.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

Cannot read block attribute separator

ACTIVE-CAD cannot read the block separator in the netlist. This error is most often caused by bad hardware and netlist file corruption.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

Cannot read signal attribute

ACTIVE-CAD cannot read signal attributes from the netlist. This error is most often caused by bad hardware and netlist file corruption.

ACTION: *Check the hardware (disk). Generate a new design netlist format.*

Appendix E

ISA Bus Simulation Library

Introduction

If you are designing a printed circuit board that goes into a PC, you need to verify it with the PC mother board. With the todays clock speeds, you cannot take any chances even if the board seems to function properly.

ACTIVE-CAD CAE tools allow you to test your designs in conjunction with PC mother boards. Equipped with the ISA bus model they allow you to test your design over the full spectrum of functionality and propagation delays. The ISA bus model also allows you to verify mother board designs over the full range of ISA specifications.

The ISA model is comprised of two kinds of modules: checker and programmable ISA bus agents. The checker monitors all bus signals to and from the ISA and warns of any violations. The programmable bus agents allow direct programming of ISA bus transactions and complete verification of the ISA interface timing.

The ISA bus model agents can be drawn on the schematic and interconnected directly to the appropriate buses and control lines on your schematic designs. The ISA bus model agents look like other IC devices on your schematic and behave like them. By selecting , drawing and programming the appropriate ISA interface, you can test your design quickly and effectively.

Each agent emulates a device directly interfacing to the ISA Bus. It is required only for simulation purposes. Each agent works independently and its operation is programmable. A memory expansion board and graphics controller are examples of such agents.

Lexicon:

The following is a description of terms used in this Appendix:

An **Agent** is a physical unit which has an interface to the Intel ISA Bus.

A **Requesting Agent** initiates an Intel ISA Bus cycle. It is either a Primary or a Secondary Requesting Agent (PRA or SRA).

A **Primary Requesting Agent** (PRA) is a required agent of which there can be only one. The only PRA is a motherboard itself.

A **Secondary Requesting Agent** (SRA) is an optional requesting agent that normally does not have immediate control of the ISA Bus. Control is requested from the PRA. Multiple SRAs are allowed. An SRA must have a 16-bit bus interface.

A **Replying Agent** (RPA) is an agent which responds to ISA Bus cycles initiated by a Requesting Agent. It cannot initiate Intel ISA Bus cycles by itself. A PRA can have an 8-bit or a 16-bit bus interface.

General

The ISA bus library is provided for verification of the design interface to the ISA bus according to the Intel ISA Bus Specification, 458057-001 Rev 1.1, January 27, 1989". The following are the modules that can be used with ISA designs:

ISA-CHECKER is a module that monitors bus signals, automatically detects all discrepancies with the Intel ISA Bus Specification and generates messages whenever a timing violation has been detected. This module is very similar to a logic analyzer that is programmed to detect specific timing problems.

ISA-PRA, ISA-SRA, ISA-RPA are programmable ISA bus agents. They allow you to effectively test your design with the environment in which your design will work. For example, if you are designing an 8-bit I/O board for the PC computer, you can add some other modules that emulate the motherboard and other 16- or 8-bit cards. By programming each bus agent to perform a sequence of I/O cycles, you can test whether your board is in conflict with the other boards in the computer and conforms to the ISA specification. Each bus agent is programmable separately and the test sequence can be saved and loaded from a disk file.

The bus agents generate bus cycles synchronized with the bus clock. Each agent can also respond to the cycles generated by other bus agents.

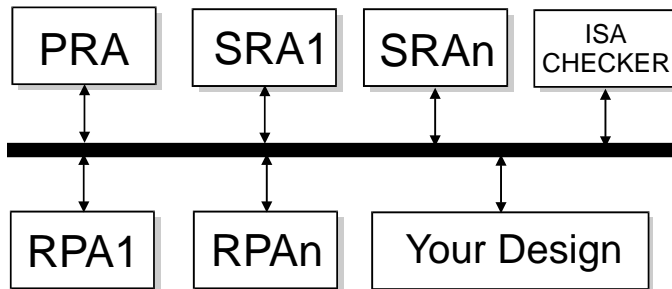


Figure E-1. ISA Bus Testing Environment.

You can place on your schematic any combination of ISA modules. Since they are used for verification only they should be deleted from the schematic after you finish the testing. To get the best results, you need to program the testing sequence, one bus agent at a time. Other agents will respond to this agents cycles. If you program more than one agent to operate at the same time, they will most likely conflict with each other.

Each agent is able to:

- ☐ respond to bus cycles according to the ISA bus specification
- ☐ generate bus cycles according to the sequence stored in its memory (RAM). These cycles are synchronous with the bus clock.

Inserting Bus agents on the schematic

To test the interface between boards, you can place any number of ISA agents on the schematic. The symbols of ISA agents can be found in the ISA library, which has to be added to the Project Library (see Chapter 1 for details on selecting Project Libraries). You can place the ISA agents on the same schematic as your design or on separate sheets. The last method is better because it allows you to quickly disconnect the test circuits from the actual design by removing the extra sheets from the current project. These ISA-related schematic sheets can also be reused in other designs. The ISA agents are placed on the schematic for simulation purposes only and should be removed before creating a PCB netlist.

Viewing agents' Registers

After loading your design into the simulator, you can open separate windows for each agent placed on the schematic. To do that, select the **Internal State Editor** option from the **Utilities** menu. The **Select Chip for Internal State Editor** window will appear with a list of devices that have special configuration windows, such as microprocessors, ISA agents, etc. Select all of the agents and open their ISE (Internal State Editor) windows.

An agents windows allows you to:

- ☐ view and edit the agents status and control registers
- ☐ enter the simulation program (list of bus cycles)
- ☐ view the agents simulation progress

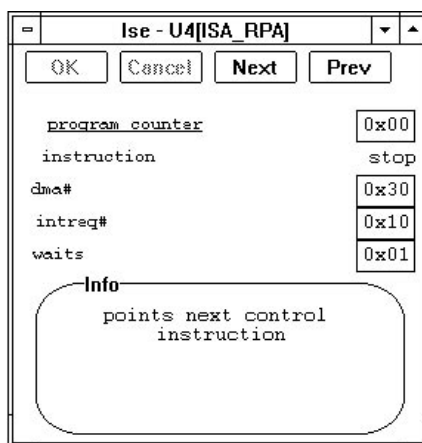


Figure E-2. ISA Agent Internal State Editor Window.

Each agents window has an **Info** section that describes the selected items, and the **Prev** and **Next** buttons that switch between different pages of the window (some agents have as many as five different pages of information.)

Editing Control Registers

To edit a control register, e.g. **dma#** in Figure E-2, click on the registers current value shown in the little box, edit it and then click on the **OK** button. This will write the new value of the register into its simulation model.

Editing the control registers is very useful if you want to specify the address or data value that should appear on the bus in the simulated cycle.

You can edit these registers interactively during the simulation and set different values for different cycles.

Editing agents simulation program

Each bus agent has a 16-instruction buffer for storing the simulation program. This program can be edited using the agents ISE (Internal State Editor) window. It can also be saved and loaded using the **Save/Load Memory Chip** options from the **File** menu. The simulation program is represented as a 256-byte memory block in the agents model.

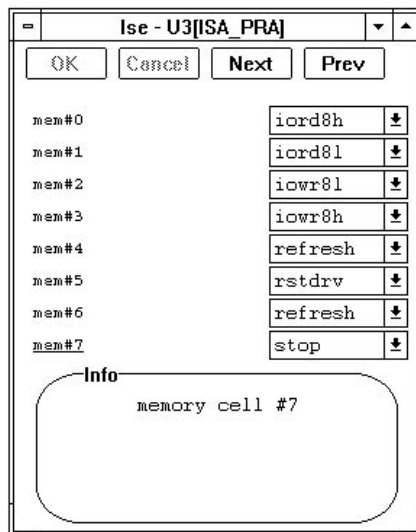


Figure E-3. Editing the Simulation Program.

To edit the program listed in the agents window use the Next button to select a page similar to Figure E-3. Then click on the desired instruction (e.g. memory #0) and select the desired cycle from the list. This list shows all available cycles for the selected agent. If you want the agent not to perform any cycles, then select the STOP instruction.

ISA Checker

The ISA-CHCK module monitors bus signals and is invisible to the rest of the devices connected to the bus. All of its pins are inputs and the only output that the module generates are error messages. ISA-CHCK is able to recognize any bus cycle and verify the timing relationship between the bus signals. It also detects logical states that are prohibited (e.g. Low

state on high address bits during the refresh cycle). All errors are reported as Timing Violations and are included in the error report.

The RESETDRV, nIOCHCK and IRQ signals are not monitored by the ISA-CHCK module.

ISA-CHCK verifies the following bus sequences:

- ☐ Memory Read
- ☐ Memory Write
- ☐ I/O Read
- ☐ I/O Write,
- ☐ DMA Read
- ☐ DMA Write
- ☐ Global Refresh
- ☐ Refresh Initiation by SRA
- ☐ Bus Arbitration and Ownership

ISA Primary Requesting Agent

The ISA-PRA is the Primary Requesting Agent, which is a default bus owner. It simulates all of the bus cycles listed above and responds to:

- ☐ Interrupt Requests,
- ☐ DMA Requests,
- ☐ Bus Arbitration Requests.

ISA-PRA has built-in signal generators for CLK and OSC that are used by all other ISA agents. The bus cycle generator works off the CLK frequency.

ISA-PRA has 256 bytes of RAM memory for storing a simulation program. In addition, ISA-PRA has status and control registers for monitoring the simulation process.

The ISA-PRA agent executes the simulation program instructions (cycles), and after each instruction it checks (during CLK transition) for the bus request (DRQi lines). If there are any requests, they are handled according to the request priority (DRQ0 first... DRQ7 last). The request handling procedure depends upon DMA mode. If there is no request on the bus, ISA-PRA executes the next program instruction until STOP. IRQ requests and nIOCHCK are handled asynchronously, immediately after they occur.

Control and Status Registers

Definitions

InstCount

An 8-bit address register pointing at the next control instruction that will be fetched from the agents simulation program memory.

CurrInst

An 8-bit register containing the current simulation program instruction.

INADDR

A 24-bit address for all Read operations (Memory Read, I/O Read, DMA Read, Refresh). This address is set by PRA on the LA/SA signal lines. The most significant bits (MSB 23..20) decide whether to set either the nMEMR or nSMEMR bus lines. For I/O Read bits (23..16) are ignored and the Low state should be set on all of these lines. Similarly during the Refresh cycle, bits 10 and up are set to Low.

INDATA

A 16-bit data register for all Read operations (Memory Read, I/O Read, DMA Read). PRA reads data from SD lines.

OUTADDR

A 24-bit address for all Write operations Memory Write, I/O Write, DMA Write). This address is set by PRA on LA/SA signal lines. The most significant bits (MSB 23..20) determine the setting of either nMEMW or nSMEMW bus lines. Bits (23..16) are ignored during I/O Write and the Low state is set on all of these lines.

OUTDATA

A 16-bit data register for all Write operations (Memory Write, I/O Write, DMA Write.) PRA sets the data on the SD lines.

DMAMode

An 8-bit register in which the user determines the response of PRA to DRQ_i bus access request. Its default value is set to SRA_SERVICE=00h:

☐ **SRA_SERVICE (00h)**

PRA releases the bus to the SRA, waiting for the nMASTER signal;

☐ **DMAMEMRD8L (01h)**

PRA generates the Read cycle of the lower data byte into the memory; INADDR determines the address that is being set; the read byte SD(7..0) is written into the least significant byte of INDATA,

❑ **DMAMEMRD16 (02h)**

PRA generates the Memory Read cycle of the entire data word; INADDR determines the address that is being set on the bus, the data word SD(15..0) is written into INDATA,

❑ **DMAMEMWR8L (03h)**

PRA generates the Memory Write cycle of the low data byte; OUTADDR determines the address set on LA/SA, the low byte of OUTDATA is stored in the byte SD(7..0)

❑ **DMAMEMWR16 (04h)**

PRA generates the Memory Write cycle of the entire data word; OUTADDR determines the address set on LA/SA, the word OUTDATA is loaded onto the SD(15..0) lines,

INTReq

16-bit register reflects the state of the IRQi interrupt requests on the ISA lines.

Bus Cycles generated by ISA-PRA

The ISA-PRA simulation instruction set consists of the following:

STOP 00h

Causes the timing generation to stop; during the idle state the program memory contains only zeroes, i.e. only STOP instructions.

MEMRD8L 01h

Generates the Memory Read cycle of the lower data byte (connected to the ISA bus). The address is put into INADDR, the value of the byte SD(7..0) is put into the lower byte of INDATA.

MEMRD8H 02h

Generates the Memory Read cycle of the higher data byte (connected to the ISA bus). The address is put into INADDR, the value of the SD(15..8) is set into the higher byte of INDATA.

MEMRD16 03h

Generates the Memory Read cycle. The address is put into INADDR, the value of the read word SD(15..0) is put into the INDATA input data register.

IORD8L 04h

Generates the I/O Read cycle of the lower data byte (connected to the ISA bus). The address is put into INADDR, the value of the read byte

SD(7..0) is put into the least significant byte of INDATA, the 16 least significant bits of INADDR are sent to the device, the most significant bits are set as Low state.

IORD8H 05h

Generates the I/O Read cycle of the higher data byte. The address is loaded into INADDR, the value of the read byte SD(15..8) is put into the higher byte of INDATA, the 16 least significant bits of INADDR are sent to the device, the most significant bits are set to the Low state.

IORD16 06h

Generates the I/O Read cycle of the entire word of data. The address is put into INADDR, the value of the read word SD(15..0) is put into the INDATA input data register, the 16 least significant bits of INADDR are sent to the device, the most significant bits are set to the Low state.

MEMWR8L 07h

Generates the Memory Write cycle of the lower byte of OUTDATA. The address is determined by OUTADDR; the transfer is made using SD(7..0)

MEMWR8H 08h

Generates the Memory Write cycle of the higher byte of OUTDATA. The address is determined by OUTADDR; the transfer is made using SD(15..8).

MEMWR16 09h

Generates the Memory Write cycle of the entire OUTDATA word. The address is determined by OUTADDR; the transfer is made using SD(15..0).

IOWR8L 0Ah

Generates the I/O Write cycle of the lower byte of OUTDATA into the I/O device (connected to the ISA bus). The address is determined by OUTADDR, the transfer is made using SD(7..0), the 16 lower bits of INADDR are sent to the device, the most significant bits are set to the Low state.

IOWR8H 0Bh

Generates the I/O Write cycle of the higher byte of OUTDATA into the I/O device (connected to the ISA bus). The address is determined by OUTADDR, the transfer is made using SD(15..8), the 16 lower bits of INADDR are sent to the device, the most significant bits are set to a Low state.

IOWR16 0Ch

Generates the I/O Write cycle of the entire OUTDATA word into the

I/O device (connected to the ISA bus). The address is determined by OUTADDR, the transfer is made using SD(15..0), the 16 lower bits of INADDR are sent to the device, the most significant bits are sent as a Low state.

REFRESH 0Dh

Generates the memory refresh cycle; address is determined by NADDR, only the lower bits SA(9..0) are relevant. If memory requires an 8-bit refresh address, the user must set bits 9 and 8 of the INADDR address register to zero.

RSTDRV 0Eh

Generates RESETDRV, - the signal which resets the bus for a period of one clock cycle.

IDLEINSTR 0Fh

This is an empty instruction. It is used to generate delay time of one clock cycle (CLK). IDLEINSTR may be inserted any number of times between other instructions.

ISA-PRA served cycles and sequences:

The model responds to the cycles generated by other devices (i.e. standard generators or user modules) as follows:

DMA Requests (DMAMode= DMAMEMRD8L, DMAMEMRD16, DMAMEMWR8L, DMAMEMWR16); the preprogrammed cycle is generated periodically until the DMA request (DRQi) is off.

Bus Arbitration Requests (DMAMode= SRA_SERVICE); the bus is released to SRA (default),

Interrupt Requests; each appearance and disappearance of an interrupt request signal is reflected accordingly in the INTReq register bits. Note: according to the standard, not all bits of this register are used; interrupt requests are registered asynchronously.

IOCHECK detection; the occurrence of an active error signal on the nIOCHCK bus stops the bus action for one clock cycle and displays an error message.

INTERNAL STATE EDITOR (ISE) enables display and on-line modification of the key model parameters and the first 16 bytes of the memory containing the simulation program. The following is a description of the displayed parameters:

PROGRAM COUNTER displays the hex value of **InstCount** parameter; the user can modify it freely to control the simulation progress. After POWER ON, InstCount=0.

INSTRUCTION displays **CurrInst** parameters as mnemonic codes; it is the last fetched program instruction; Cannot be edited by the user.

DMAmode displays the DMAmode parameter with mnemonic codes; the user may modify it freely to control the ISA-PRA reaction to the DRQi requests; after POWER ON, DMAmode =SRA_SERVICE,

INTReq displays the binary format of INTReq parameter; each bit reflects one line of the IRQi bus; non-existing lines (as determined by the standard) are displayed as asterisks (e.g. *111*000),

INADDRH, INADDRM, INADDRL display each byte of the input address in the hex format. You can set this address to a desired value.

INDATAH, INDATAL display each byte of the input data in hex format. You can modify these registers.

OUTADDRH, OUTADDRM, OUTADDRL display each byte of the output address in the hex format, which the user can modify.

OUTDATAH, OUTDATAL display each byte of the output data in the hex format, which the user can modify.

MEM#0..MEM#15 display the first 16 simulation program instructions; each cell is displayed in mnemonics; modifications are carried out by using the list box. The program memory can be loaded and saved in a hex file.

ISA Secondary Requesting Agent

ISA-SRA is the Secondary Requesting Agent representation. SRA can generate data transfer cycles on the bus if PRA gives it permission to control the bus. SRA requests bus access by setting DRQi. In reply PRA sets DACKi and releases the bus while the SRA sets nMASTER and begins the data transfer managed by itself (i.e. SRA). After completing the transfer, SRA returns the bus control to PRA by setting the nMASTER signal to High. While having control over the ISA bus SRA is responsible for refreshing dynamic memory in the system. If SRA does not perform this task by itself, it initiates the refresh process by releasing the bus management to PRA for short periods of time.

The SRA model is clocked by the CLK bus signal. The ISA-SRA has 256 bytes of memory for storing program instructions. There is also a set of status and control registers. After completing execution of each instruction (bus cycle), the next instruction is fetched from the memory (assuming it is not STOP) during the CLK transition from Low to High.

Note: The ISA-SRA cannot distinguish between communication with DRAM and communication with other memories (timing values are not appropriate for DRAM.)

Following is a description of the status and control variables:

InstCount

An 8-bit address register pointing to the control instruction which will be fetched from the memory after executing the current instruction.

CurrInst

An 8-bit register containing the currently executed cycle.

BusOwner

A Boolean value which shows whether PRA currently controls the bus.

INADDR

A 24-bit address for all Read operations (Memory Read, I/O Read, DMA Read, Refresh). This address is set by SRA on the LA/SA signal lines. The most significant bits (MSB 23..20) decide about setting of either the nMEMR or nSMEMR bus lines. For I/O Read, bits (23..16) are ignored and the Low state is set on all of these lines. Similarly, during the Refresh cycle bits 10 and up are set to Low.

INDATA

A 16-bit data register for all Read operations (Memory Read, I/O Read, DMA Read); SRA reads the data from the SD lines.

OUTADDR

A 24-bit address for all Write operations (Memory Write, Write I/O). This address is set by SRA on the LA/SA signal lines. The most significant bits (MSB 23..20) decide about setting of either the nMEMW or nSMEMW bus lines. For I/O Write, bits (23..16) are ignored and the Low state is set on all of these lines.

OUTDATA

A 16-bit data register for all output operations (Memory Write, I/O Write); SRA sets the data on the SD lines.

DMA_NO

An 8-bit register in which the user specifies the unique number of the access request line (DRQi).

Note: According to the standard, some of the lines represent 8-bit channels; the DRQ4 line does not exist. After POWER ON, DMA_NO=0:

INT_NO

An 8-bit register in which the user specifies a unique number for the interrupt request line (IRQi).

Note: According to the standard, some lines do not physically exist, after POWERON INT_NO=3:

Cycles generated by ISA-SRA

Following is a description of the ISA-SRA simulation instruction set:

STOP 00h

Instruction stops the timing generation; during the idle state the program memory contains only zeroes, i.e. only STOP instructions.

SRAMEMRD 01h

Generates the Memory Word Read cycle. The address is put into INADDR, the value of the read word SD(15..0) is put into the INDATA input data register.

SRAIORD 02h

Generates the I/O Word Write cycle from the OUTDATA data register. The address is determined by OUTADDR, the transfer is made by using SD(15..0).

SRAMEMWR 03h

Generates the Memory Word Write cycle from the OUTDATA data register. The address is determined by OUTADDR, the transfer is made using SD(15..0).

SRAIOWR 04h

Generates the I/O Word Write cycle from the OUTDATA data register. The address is determined by OUTADDR; the transfer is made using SD(15..0).

BUSREQ 05h

Generates the Bus Arbitration sequence by setting DRQi (specified by the user in the control word DMA_NO) After gaining control over the bus,

SRA can execute any combination of data transfer cycles. SRAMEMRD, SRAIORD, SRAMEMWR and SRAIOWR initiate the DRAM refreshing cycle, and then they release the bus to PRA using RELBUS.

RELBUS 06h

Generates the Release Bus sequence by setting off the nMASTER signal, thus returning control of the bus back to PRA.

INTSET 07h

Sets the IRQ_i interrupt request signal to on (IRQ number is defined by the user in the control word INT_NO); the interrupts are handled by PRA. The standard does not force the way the interrupts are handled.

INTRST 08h

Sets the IRQ_i interrupt request signal to off.

REFRINIT 09h

Initiates the DRAM refreshing cycle. SRA activates the nREFRESH signal. PRA generates the refresh cycle and prolongs the duration time of the nREFRESH signal. Disappearance of the nREFRESH signal on the bus completes the execution of the REFRINIT instruction,

IDLEINSTR 0Ah

This is an empty instruction. It is used to generate delay time of one clock cycle (CLK). IDLEINSTR may be inserted any number of times between other instructions.

The number of cycles that SRA can perform depends on whether it currently controls the bus. The BusOwner register value is true when SRA controls the ISA bus. BusOwner is set to TRUE by BUSREQ and set to FALSE by RELBUS.

The following instructions are permitted while BusOwner=TRUE:

- ☐ STOP
- ☐ RELBUS
- ☐ SRAMEMRD
- ☐ INTSET
- ☐ SRAIORD
- ☐ INTRST
- ☐ SRAMEMWR
- ☐ REFRINIT
- ☐ SRAIOWR

These instructions are permitted while BusOwner=FALSE:

- ☐ STOP
- ☐ INTSET
- ☐ BUSREQ
- ☐ INTRST

ISA-SRA served cycles and sequences

The model responds to the cycles generated by other devices (i.e. stand-ard generators or user modules) as follows:

RESETDRV detection; the occurrence of an active RESETDRV signal generates a message.

INTERNAL STATE EDITOR (ISE) enables display and on-line modifica-tion of key model parameters and the first 16 bytes of the memory con-taining simulation program instructions. The following is a description of the display registers:

PROGRAM COUNTER displays the value of the of InstCount param-e-ter; the user can modify it to control the program execution; after POWER ON, InstCount=0.

INSTRUCTION displays CurrInst parameter in mnemonic codes; it is the last fetched instruction; Cannot be edited by the user.

BUS OWNER displays if the bus is controlled by PRA or SRA: PRA(Bus-Owner=FALSE) or SRA (BusOwner=TRUE).

DMA# displays DMA_NO parameter.

INTREQ# displays INT_NO parameter.

INADDRH, INADDRM, INADDRL display each byte of the input ad-dress in the hex format, which the user can modify.

INDATAH, INDATAL display each byte of the input data in the hex for-mat, which the user can modify.

OUTADDRH, OUTADDRM, OUTADDRL display each byte of the out-put address in the hex format, which the user can modify.

OUTDATAH, OUTDATAL display each byte of the output data in the hex format, which the user can modify.

MEM#0..MEM#15 display the first 16 simulation program instructions; each cell is displayed in mnemonics. Modification is possible by using the list box with available instructions. The program can be saved and loaded from a hex file.

ISA Replaying Agent

ISA-RPA is the Replaying Agent. RPA is controlled by other modules (PRA or one of the SRA agents) connected to the bus. ISA-RPA requests a service by the following sequence:

- ☐ Interrupt Requests,
- ☐ DMA Requests,

The RPA responds to the following sequences generated by the PRA or SRA:

- ☐ Memory Read
- ☐ Memory Write
- ☐ I/O Read
- ☐ I/O Write
- ☐ DMA Read
- ☐ DMA Write
- ☐ Global Refresh

The model generates its cycles according to the CLK frequency on the bus. ISA-RPA has a 256 byte RAM memory for storing program instructions. In addition, there is a set of status and control registers used to store these parameters; the registers are accessible through the INTERNAL STATE EDITOR windows.

InstCount

An 8-bit address register pointing at the next program instruction that will be fetched from memory after completing the execution of the current instruction.

CurrInst

An 8-bit register containing the currently executed instruction.

INADDR

A 24-bit address for all input operations (Write cycles into the ISA-RPA:Memory Write, I/O Write, DMA Write). ISA-RPA may be treated as a memory or I/O device. ISA-RPA compares the LA/SA address with an address in the INADDR register. If they are identical the ISA-RPA module is activated.

INDATA

A 16-bit data register for data read from the SD lines into ISA-RPA.

OUTADDR

A 24-bit address for all output operations (the Read cycles from the ISA-RPA Memory Read, I/O Read, DMA Read). ISA-RPA may be treated as a memory or I/O device. ISA-RPA compares the LA/SA address with an address in the OUTADD Register. If they are identical, the ISA-RPA module becomes active.

OUTDATA

A 16-bit data register for all output operations for the data output from the ISA-RPA to SD lines.

DMA_NO

An 8-bit register in which the user specifies the number of the bus access request line(DRQi).

Note: According to the standard, some of the lines represent 8-bit channels; DRQ4 line does not exist. After POWER ON, DMA_NO=1.

INT_NO

An 8-bit register in which the user specifies the number of the interrupt request line (IRQi).

Note: According to the standard some of the lines do not physically exist, after POWERON INT_NO=3.

WAITS

An 8-bit register where you can specify the number of WAIT cycles, (defines the time when the Ready signal is set during the I/O RPA cycle). n0WS is set only if WAITS=0. nIOCHRDY is delayed by one CLK cycle when WAITS=0 or 1. It is delayed by X CLK cycles when WAITS=X (X=2 and more) after POWER ON WAITS=1.

Cycles generated by ISA-RPA

STOP	00h
INTSET	01h
INTRST	02h
DMASET	03h
DMARST	04h
IOCHCKSET	05h
IOCHCKRST	06h
IDLEINSTR	07h

INTERNAL STATE EDITOR (ISE) window displays the following parameters for the RPA agent:

PROGRAM COUNTER displays the value of the InstCount parameter; the user can modify it to control the simulation program execution. After POWER ON, InstCount=0.

INSTRUCTION displays the CurrInst parameter as a mnemonic code; it is the last fetched simulation instruction. It cannot be edited by the user.

BUS OWNER displays the BusOwner parameter: PRA (BusOwner=FALSE) or SRA (BusOwner=TRUE).

DMA# displays DMA_NO parameter.

INTREQ# displays INT_NO parameter.

INADDRH, INADDRM, INADDRL display each byte of the input address in the hex format, which the user can modify.

WAITS displays the number of WAITS.

MEM#0..MEM#15 display the first 16 simulation instructions; each cell is displayed in mnemonics. Modification is feasible by using the list box with available cycles. The program can be saved and loaded as a hex file.

Examples

The set of bus agents provided in the ISA library allows you to design a sophisticated test configuration for verifying your designs behavior during various bus situations. The following examples demonstrate a few applications of the ISA models.

Example 1:

Your design is a multiport I/O card (RPA type). To verify the ISA bus interface place the following modules on your schematic or separate sheets and connect them to the ISA signals in your design:

- ☐ ISA-PRA to emulate the motherboard
- ☐ ISA-CHK to monitor the bus timing

After loading the design into the simulator, select the PRA agents window (**Internal State Editor** option within the **Utilities** menu). Enter the simulation program by selecting the desired bus cycles in the Memory

registers #0 through #0Fh. Select the appropriate values for the address and data control registers. Put the desired signals to the Waveform Viewer. Generally, you do not need to design any test vectors because the PRA agent generates the CLK and OSC signals. The frequency of these signals is specified by TcC and TcOSC timing parameters, respectively. To edit these values, use the **Edit Timing Specification** option from the **Patching** menu.

The simulation starts with the Power On message window. Click the OK button. The first simulation step can be lengthy since the system calculates all initial design states. From that point, the PRA agent controls the design cycles according to the simulation program and will repeat the entire sequence after reaching the last cycle. Remember that the agents windows allow you to edit only the first 16 instructions. The remaining part of the 256 instructions have to be edited as a memory or loaded from a hex file.

Example 2:

You design a secondary processor board (SRA type), that works in parallel with the main processor. To verify the ISA bus interface, place the following modules on your schematic:

- ☐ ISA-PRA to generate main board signals
- ☐ ISA-SRA (one or more) to verify bus arbitration
- ☐ ISA-CHK to verify the bus timing

The programmed ISA-PRA will test the data exchange with the designed module. Edit the simulation program of the PRA agent so that it generates the transfer cycles handled by your board. Remember to correctly set the control registers (data, address, etc.) Then start the simulation.

The arbitration is tested when the SRA agents, other than yours, are present.

Example 3:

If you design the main PC board (PRA type), place the following modules on your schematics:

- ☐ ISA-SRA (one or more) to verify the arbitration
- ☐ ISA-RPA (one or more) to verify data exchange
- ☐ ISA-CHK to verify bus timings

In this case, your design controls the ISA bus and by programming different SRA and RPA agents you can test all possible cycles and configura-

tions. You can program more than one agent at a time, but make sure that you do not generate conflicts between each other. To control cycles generated by multiple agents, use the IDLEINSTR (Idle Instructions) so that only one agent module generates the cycles at any given time.

INDEX

A

Active output pin 1-18
 Add New Comment window 2-58
 Add Signals option 1-7, 1-9, 2-23
 Add Stimulators option 1-14 - 1-15, 2-8
 Advantages of simulation 1-12
 ALDEC ASCII netlist format B-1 - B-7
 ALDEC macro operations 5-29 - 5-40
 Check Design 5-30
 Check MemBlock 5-30
 Check MemChip 5-31
 Check Timing 5-31
 Get Netlist 5-32 - 5-33
 Load ASCII 5-33
 Load Design 5-33 - 5-34
 Load Fault 5-34
 Load Fuse Map 5-34
 Load Memory Block 5-35
 Load Memory Chip 5-35
 Load Selective Preset 5-35
 Load Timing 5-36
 Save ASCII 5-36
 Save Design 5-36 - 5-37
 Save Fault Timing 5-37
 Save Memory Block 5-37
 Save Memory Chip 5-37 - 5-38
 Save Selective Preset 5-38
 Save Timing 5-38 - 5-39
 ALDEC netlist structure B-1
 #COMP section B-3 - B-4
 #CONF section B-4 - B-5
 #ENDN section B-7
 #FILE section B-2
 #NODE section B-6
 #QNTY section B-2 - B-3
 #SIGN section B-5 - B-6
 #TYPE section B-3
 List of special characters B-2
 Alignment between signals 1-28
 Analyzing simulation results 1-26 - 1-33, 2-53, 2-55, 2-57, 2-59
 Analyzing waveform displays 2-53 - 2-54
 Applying signal waveforms 1-23 - 1-24, 1-39 - 1-40
 ASCII test vector files 2-62

ASCII Timing Blocks A-1
 ASCII Timing File Structure A-1
 ASCII Timing Format Specification A-1 - A-4

#End A-4
 #Hier block A-1
 #Signals block A-2
 #Timing block A-3
 ASCII timing file structure A-1
 Example A-4

ASICs

Setup and hold times 1-49
 Assign Formula button 1-21
 Assign Netlist option 1-79

B

Backup
 Automatic 2-9
 Bc 2-33
 Bi-directional terminal B-6
 Binary counter 1-18 - 1-19, 2-34
 Clock 1-19
 Outputs 1-19
 Block, #COMP B-3
 Block, #CONF B-4
 Block, #ENDN B-7
 Block, #FILE B-2
 Block, #Hier A-1
 Block, #NODE B-6
 Block, #QNTY B-2
 Block, #TYPE B-3
 Blocks, ASCII Timing A-1
 Boolean equations 4-1
 Breadboard, electronic
 Creating 1-3 - 1-7
 Breakpoint conditions 6-2 - 6-3
 Buses 6-4
 Individual signal 6-3
 Breakpoint editor 6-2 - 6-3, 6-5 - 6-7
 Breakpoint options menu 6-1
 Breakpoints 2-11, 2-71, 5-9 - 5-10, 6-1 - 6-7
 Event format 5-10
 Bus
 Breakpoint conditions 6-4
 Conflicts 1-31
 Creating 1-34 - 1-35, 2-8, 3-4
 Define in Simulator 2-27 - 2-28
 Define in Test Editor 3-4

- Deselect 2-29
- Destroy 2-29
- Display 1-35, 2-8
- Enter current value 3-6
- Expand 3-4
- Logical states 5-3 - 5-4
- Members 3-3
- Naming, renaming 1-36, 3-3
- Segment duration 3-6
- Sub-menu 2-28 - 2-29
- Working with 1-34 - 1-36
- Bus agents
 - Inserting E-3
 - Programmable E-1 - E-7, E-16, E-18 - E-20
 - Viewing registers E-4
- Bus Conflicts 2-62
- Bus cycles generated by ISA-PRA E-8 - E-10
- Bus names
 - Entering 3-3
- C**
 - Case Sensitive B-2
 - Change Generic Values 2-10, 2-67
 - Change Line Delays 2-10
 - Change Line Delays option 1-66, 2-65
 - Change technology 2-10, 2-66
 - Chip declaration 4-2
 - Chip-level
 - Control 2-8
 - Simulation 1-78
 - Chip-selection field 1-33, 2-26
 - Clock button 2-40
 - Clock Editor 1-16
 - Clock Settings option 1-19, 2-9, 2-34
 - Clock stimulators 1-15 - 1-16
 - Clocks 2-33
 - Applying 1-71
 - Asynchronous 2-39
 - Binary counter 1-18 - 1-19
 - Creating 1-71
 - Multiple 1-70
 - Colors 2-55
 - Comments
 - Inserting 2-58
 - Comparison with other simulators ii-ix
 - Compiling a model 4-4
 - Component names 5-2
 - Component Selection window 1-6 - 1-8, 1-11, 2-23, 2-26
 - Components
 - Locating 1-37 - 1-38
 - Passive 1-69 - 1-70
 - Connections
 - View 2-30
 - Connections option 1-76, 1-78, 2-30
 - Connections window 2-30
 - Connectivity
 - Design 1-36 - 1-37, 1-73
 - Pins 1-6
 - Signal 1-36, 2-8 - 2-9
 - Control and status registers E-7 - E-8
 - Creating a bus 1-34
 - Creating reports 2-63
 - Cross-probing 1-38
 - CS symbol 1-16, 1-23
 - Cycles generated by ISA-SRA E-13 - E-15
- D**
 - Debugger
 - VHDL 2-11
 - Defined Assignments field 1-21
 - Delete stimulus signals 2-9
 - Deleting empty rows 1-34
 - Design analysis 1-12 - 1-13
 - Design connectivity 1-36 - 1-37
 - Design environment 1-2
 - Design error reporting 2-63
 - Design initialization 2-43
 - Design macros
 - Disabled 2-51
 - Enabled 2-51
 - Design netlist 1-4, 1-9, 1-12, 1-73
 - Design preset
 - Manual 1-68
 - Directories window 1-5 - 1-6
 - Disabled design sections 1-53
 - Disabled device 1-53, 2-52
 - Disconnect 2-9
 - Display scale 1-28
- E**
 - EDIF B-1
 - Edit
 - Agent simulation program E-5
 - Control registers E-4 - E-5
 - Propagation delays 2-64 - 2-65
 - Timing parameters 2-65
 - Timing specification 2-10, 2-65, 2-67

- Waveforms 1-21 - 1-23, 2-9, 2-36, 2-41
- Editor
 - Internal state 2-11
 - Memory 2-11
 - Signal waveform 1-14
 - Test vector (external) 1-14
 - Test vector macro 1-14
- Editor buffer
 - Contents 3-4
- Effective design testing 1-12
- Electronic breadboard 1-1 - 1-3, 1-7
 - Creating 1-3 - 1-6
- Empty line
 - Delete 2-72
 - Insert 2-8
- END statement 4-2, 4-13
- EQU statment 4-2, 4-12
- Error Messages D-1 - D-9
- Error option 2-54
- Error Reports 2-62 - 2-63
 - Bus conflicts 2-62
 - Create 2-63
 - Timing violations 2-62
- Errors
 - Classes 1-46
 - Correcting design timing 1-46 - 1-47
 - Handling 1-46
 - Line delays 1-51
 - Reporting 1-45 - 1-46, 2-10, 2-62 - 2-63
 - Setup time 2-62
 - Timing violations, device-related 1-50 - 1-51
 - Tracking through a netlist 1-75 - 1-78
 - Viewer 2-11, 2-63
- Event files 5-7
- Example
 - Test vector file 5-6 - 5-7
- Expanding Scale Resolution button 1-28
- External Netlist Format window 1-73
- External Text Vector Editor 3-1, 3-3 - 3-7
- F**
- Fault simulator 2-11
- File menu 2-7, 2-15
 - Load 2-7
 - Load ASCII Test Vectors 2-7, 2-40, 2-62
 - Load Fuse Map 2-7, 2-17 - 2-18
 - Load Memory Block 2-7
 - Load Memory Chip 2-7, 2-20 - 2-21
 - Load Netlist 2-8
 - Load Simulation 2-7, 2-60
 - Load Test Vectors 2-7
 - Page Setup 2-7
 - Print 2-7
 - Print Error Report 2-7
 - Print Setup 2-7, 2-73
 - Project Libraries 2-8, 2-15
 - Project Manager 2-8
 - Save ASCII Test Vectors 2-7, 2-62
 - Save Memory Block 2-7
 - Save Memory Chip 2-7
 - Save Simulation 2-7, 2-60
 - Save Test Vectors 2-7
 - Test PLD 2-8, 2-18 - 2-19
- File structure
 - Event 5-7
 - Macros 5-5 - 5-6
 - Test vectors 5-6 - 5-7
- Files, ASCII Timing A-1
- Flat netlist 1-73
- Flat netlist format 1-5, 1-9
- Flat netlist format 1-5, 1-9 1-6
- Format
 - External 2-14
- Formula editor 1-20, 2-37
- Formula stimulators 1-15, 1-19 - 1-21, 1-40, 2-33
- Formula-based signals 1-39 - 1-42, 2-9, 2-33, 3-6
- FPGA
 - Multiple 1-79
- FPGA designs
 - Simulating 1-79
 - Simulating, system level 1-78
- From Format field 1-4 - 1-5
- Functional analysis 1-27, 1-29
- Functional simulation 1-26 - 1-29, 2-49
- Functional system-level analysis 1-78
- G**
- Generic Values 2-10
 - Changing 2-67
- Glitch mode 1-29
- Glitch simulation 1-26, 1-28 - 1-31, 2-50
- Global
 - Propagation delays 1-50
 - Propagation setups 1-49
 - Reset 1-45, 1-54 - 1-57, 2-10, 2-47

Graphical waveform editor 1-21 - 1-23, 2-35 - 2-36
 Grid option 3-6
 Grid spacing 3-6

H

Hardware breadboard 1-1 - 1-3, 1-7, 1-12, 1-65
 Help menu 2-11
 Hierarchical designs 1-9 - 1-10, 1-73
 JEDEC fuse maps 2-18
 Loading HEX files 2-21
 Schematics 1-9
 Simulation 1-9, 2-26
 Hierarchical format 1-73
 Hierarchical macros 1-73
 Hierarchical netlist 1-9, 1-73, 2-13
 Hierarchical simulation option 1-9
 Hierarchical viewer 2-11
 Hierarchy
 Levels in Simulator 2-26
 Netlist 2-13
 Signals 2-8
 HIGH state terminal B-6
 High voltage terminal B-6
 Hold times
 ASICs 1-49

I

Impedance, High A-3
 Import Netlist 2-13
 Incremental netlist updating 1-72
 Inductors 2-68
 Input Netlist field 1-5
 Input pins
 Overriding 1-17
 Input terminal B-6
 Input, High A-3
 Input, Low A-3
 Inserting comments 2-58
 Internal label B-6
 Internal state editor 2-11
 ISA bus model agents E-1 - E-7, E-16, E-18 - E-20
 ISA bus simulation library E-2 - E-20
 Import messages E-1
 ISA checker E-5 - E-6
 ISA control and status registers E-7 - E-8
 ISA model E-1, E-18
 Agent E-1 - E-5, E-7

Primary requesting agent E-2, E-6
 Replying agent E-2, E-16 - E-18
 Requesting agent E-2
 Secondary requesting agent E-2, E-11 - E-13

ISA-PRA

Bus cycles generated by E-8 - E-10
 Served cycles and sequences E-10 - E-11

ISA-SRA

Cycles generated by E-13 - E-15
 Served cycles and sequences E-15 - E-16

J

JEDEC test vectors 2-20
 Jumpers
 Move 1-65
 Toggle 2-67

K

Keyboard key assigned signals 2-35
 Keyboard key controlled signals 2-34
 Keyboard keys 1-15 - 1-18, 2-33
 Keyword, #End A-4

L

Large designs 1-52
 Large memory
 Simulation 1-62
 Large memory devices
 Simulation 1-62
 Layout changes
 Emulating 1-67
 Layout propagation delays 1-66
 LF B-2
 Library Manager 2-64
 Line Delays
 Analyzing 1-51
 Change 1-66, 2-10, 2-65
 Editing 2-65
 Simulating 1-66 - 1-67
 Line Delays:Table 2-66
 Line Feed B-2
 Load
 ASCII Test Vectors 2-40, 2-62
 Design 2-12 - 2-13, 2-15 - 2-23
 Fuse map 2-7, 2-17
 HEX files 2-21
 JEDEC file into PLD 2-19
 Memory block 2-7
 Memory chip 2-7, 2-20 - 2-21

- Memory contents 2-20
- Milestones 2-71
- Netlist 2-8, 2-12 - 2-15, 2-17
- PLD fuse map 2-16
- PLD test vectors 2-19
- Simulation 2-7, 2-60
- Test vectors 2-7, 2-62
- Load Netlist option 1-4
- Load Netlist window 1-4, 1-6
- Locating
 - Components 1-37 - 1-38
 - Pins 1-37 - 1-39, 2-8
 - Signal conditions 1-62 - 1-64
 - Signal names 1-37 - 1-39, 2-8
- Logic equations 4-1
- Logical states
 - Buses 5-3 - 5-4
 - List 1-22 - 1-23
 - Representation 5-3 - 5-4
 - Select 3-5
 - Waveform signals 3-5
- Long simulations 2-53
 - Running 1-60
 - Start 1-65
 - Stop 1-65
- Long Step 2-43
- LOW state terminal B-6
- Lower Scale Resolution button 1-28
- M**
- Macro 2-11
- Macro file structure 5-5 - 5-6
- Macro naming conventions
 - Component names 5-2
 - Pin names 5-2
 - Signal names 5-2 - 5-3
- Macro operations 5-1 - 5-39
 - Logical states 5-3 - 5-4
 - Macro file structure 5-5 - 5-6
 - Naming conventions 5-2 - 5-3
 - Simulator macros 5-1
 - Time units 5-4 - 5-5
 - Viewsim compatible macros 5-7 - 5-29
- Main menus 2-7 - 2-11
 - File menu 2-7 - 2-8
 - Help menu 2-11
 - Options menu 2-9 - 2-10
 - Patching menu 2-10
 - Signal menu 2-8
 - Stimulator menu 2-8 - 2-9
 - Utilities menu 2-10 - 2-11
 - Waveform menu 2-9
 - Window menu 2-11
- Maximum propagation delay 1-48
- Measurements button 2-56
- Measuring time intervals 2-55, 3-7
- Memory
 - Configurator 2-11
 - Contents 2-22
 - Load Hex File 2-20
 - Range 2-10
 - Simulation Range 2-22
- Memory block
 - Load 2-7
 - Save 2-7
- Memory chip
 - Load 2-7
 - Save 2-7
- Memory Editor 2-11, 2-22, 2-59
- Memory Editor option 2-59
- Memory Range 2-22
- Memory Range option 1-62, 2-22
- Milestones 2-69
 - Automatic 1-58 - 1-59, 2-70
 - Breakpoint-driven 1-60, 2-70
 - Delete 2-71
 - Loading 2-71
 - Manual 1-59 - 1-60, 2-70
 - Option 1-58, 2-9
 - Saving 2-69 - 2-71
 - Selected 2-69
 - Setting 1-58
 - Time interval 1-59
- MOBIC 6.0 specification
 - Chip names 4-6
 - Class declarations 4-8
 - Instruction names 4-7 - 4-8
 - Instruction syntax 4-13
 - Logical states 4-9
 - Model structure 4-6
 - Operators 4-9
 - Pin names 4-6
 - Pin types 4-8
 - Power-On signal 4-10
 - Reserved words 4-7
 - Set of characters 4-6
 - Signal names 4-6
 - Source text 4-6

- Statement names 4-7 - 4-8
- Violation codes 4-10
- MOBIC instruction syntax
 - Conditional IF 4-14
 - Memory, Read from 4-14
 - Memory, Write to 4-15
 - Temporary signals 4-15
 - Violation, report 4-16
- MOBIC logical equations syntax
 - Equation syntax 4-17
 - Logic, 12-value 4-17
 - Memory model addressing 4-20
 - Operator priority 4-17
 - Signals and pins 4-20
- MOBIC Model Builder Compiler 4-1 - 4-24
 - Chip declaration 4-2
 - Compiling a model 4-4
 - Model Body 4-2
 - Model conventions, combinatorial 4-3
 - Model conventions, sequential 4-3
 - Pin definition 4-2
 - Signal definition 4-2
 - Source text 4-6
- MOBIC sample models
 - Gates 4-21
 - Multiplexer 4-21
- MOBIC statement syntax 4-11
- MOBIC statements
 - END 4-2
 - EQU 4-2
 - PINS 4-2
 - SIGNALS 4-2
- Model
 - Body 4-2
 - Compiling 4-4
 - Conventions, Combinatorial 4-3
 - Conventions, Sequential 4-3
 - Missing 2-15
 - Structure 4-6
- Mouse 3-2
 - Left button 3-2
 - Right button 3-2
- Moving signals 3-7
- Moving test vectors to another design 1-75
- Multiple clocks 1-70

N

- Naming buses 1-36
- NBc 2-33
- Net signal names
 - Overriding 1-18
- Netlist
 - Assign 2-13
 - Board-level 2-12
 - Combining 2-13
 - External 1-73
 - Flat, simulation 1-73
 - Format 1-3 - 1-5
 - Format, external 2-14
 - Format, flat 1-5
 - Hierarchical 1-74, 2-13
 - Hierarchical, simulation 1-73 - 1-74
 - Import 2-13
 - Import messages D-1 - D-9
 - Incremental 1-72
 - Input 1-5
 - Load 1-4, 1-6, 2-12, 2-14, 2-17
 - Off-line mode 1-73
 - Simulating 1-71, 1-73, 1-75, 1-77, 1-79
 - Simulating external 1-75
 - Tracking errors 1-75 - 1-78

O

- Off-line test vector editor 1-14
- Options menu 2-9
 - Clock Settings 2-9, 2-34
 - Default Settings 2-10, 2-74
 - Delete Tag Condition 2-10
 - End of Step Estimation 2-9
 - Error Reporting 2-10, 2-62 - 2-63
 - Global Reset 2-10
 - Memory Range 2-10, 2-22
 - Milestones 2-9, 2-71
 - Power On Settings 2-9, 2-44
 - Save Settings Now 2-10, 2-73
 - Save Settings on Exit 2-10, 2-74
 - Selective Simulation 2-9, 2-51
 - Set Tag Condition 2-10, 2-57
 - Simulation Precision 2-9
 - Simulation Stop 2-9, 2-53
 - Timing Automatic Backup 2-9
 - Transport Delay 2-10
- Output pins
 - Overriding 1-18
- Output signals

- Unknown 2-43
- Output terminal B-6
- Output, High A-3
- Output, Low A-3
- Overdrive mode 1-24
- Override
 - Input pins 1-17
 - Mode 2-8
 - Net signal names 1-18
 - Output pins 1-18
- Override Mode 1-18, 2-23 - 1-24

P

- Page Setup
 - Simulator 2-7, 2-71 - 2-72
- Part
 - Replaced 2-67
- Passive Components 1-69, 2-68
- Patching menu 1-33, 1-48, 1-65, 2-10
 - Change Generic values 2-10, 2-67
 - Change Line Delays 2-10, 2-65
 - Change Technology 2-10
 - Edit Timing Specifications 2-10
 - Switch Settings 2-10
- Pin connectivity information 1-6
- Pin transfer
 - Individual 2-24
- Pins
 - Definition 4-2
 - Locating 1-37 - 1-39
 - Names 5-2
 - Overriding input 1-17
 - Overriding output 1-18
 - Selecting 1-8
- Pins statement 4-2
- PLD
 - Load JEDEC File 2-16
 - Loading fuse map 2-16
 - Selecting 2-19
 - Test PLD 2-8, 2-18
- Power-On 2-46, 2-48
 - Button 1-56, 2-46 - 2-47
 - Executing 1-55
 - Global reset 1-57
 - Instructions 1-56
 - Parameters 1-56
 - Preset 1-57
 - Settings 1-56, 2-9, 2-44, 2-47
 - Settings, default 1-56

- Power-On Setting window 1-56
- Precision
 - Simulation 1-42 - 1-43, 2-9, 5-5
 - Timing 5-5
- Preset 1-55, 1-57
 - Automatic 1-69
 - Design 1-67 - 1-68
 - Manual 1-69
 - Selective 2-11, 2-45
- Preset conditions
 - Forcing 2-46
 - Selecting 2-45
- Print 2-7
 - Error report 2-7
 - Options 2-72 - 2-73
 - Results 2-71 - 2-73
 - Setup 2-7, 2-73
 - Waveforms 2-71 - 2-73

- Probes
 - Select in Simulator 2-23 - 2-24
 - Select on Schematic 2-23
- Project Library 2-8, 2-12, 2-15
- Project Manager 2-8, 2-12, 2-17, 2-21
- Propagation Delays 1-49 - 1-50
 - Editing 2-64 - 2-65
 - Layout 1-66
 - Maximum 1-48
 - Rescale 1-50
 - Rescaling 1-67
 - Result of 1-30
 - Unit 2-50

R

- Race conditions 1-26, 1-28 - 1-30
- Reference voltage terminal B-6
- Renaming buses 1-36
- Report window 2-63
- Reset
 - Design 1-45, 1-54 - 1-55
 - Global 1-45, 1-54 - 1-55, 1-57, 2-10
- Resimulation 1-21
- Resistors 2-68
 - Pull-Down 2-69
 - Pull-Up 2-69
- Ruler option 3-5

S

- Sample, #End A-4
- Sample, #Hier A-4
- Sample, #Signals A-4

- Sample, #Timing A-4
- Save
 - Layout configuration 2-74
 - Memory block 2-7
 - Memory chip 2-7
 - Milestones 2-69, 2-71
 - Settings 2-10, 2-73
 - Simulation 2-7, 2-60
 - Test vectors 2-7, 2-62
 - Windows layout 2-73 - 2-74
- Save ASCII Test Vectors option 2-62
- Save selected signal names 1-74
- Save Test Vectors option 1-9
- Scale resolution 2-42
- Scan Hierarchy window 1-11, 1-52 - 1-53, 1-65
- Schematic editor
 - Off-line 2-12
 - On-line 2-12
- Search button 2-55
- Searching for tags 1-64
- Select Memory Chip window 2-20
- Select Netlist button 1-73
- Selected Stimulator field 1-21
- Selecting bus display format 1-35
- Selecting signals for display 2-23 - 2-25, 2-27, 2-29, 2-31
- Selection
 - Device pins 2-26
 - Probes 2-23
 - Signals 2-23 - 2-24
 - Test points 1-1, 1-7 - 1-12
- Selective design simulation 1-52, 1-54, 2-9
- Selective Preset 2-11, 2-44 - 2-46
- Selective Preset window 2-45 - 2-46
- Selective simulation 1-53 - 1-54, 2-9, 2-51 - 2-52
- Settings
 - Default 2-10, 2-74
 - Save 2-10, 2-74
 - Switch 2-10
- Setup times
 - ASICs 1-49
- Short Step 2-43
- Signal
 - Weak 1-69
- Signal conditions
 - Locating 1-62 - 1-64
 - Selected 1-69
- Signal connectivity 1-36
- Signal line
 - Toggling 1-17
- Signal menu 2-8
 - Add Signals 2-8, 2-23
 - Bus 2-8, 2-28
 - Connections 2-8, 2-30
 - Delete 2-8
 - Find...in SC 2-8
 - Hierarchy 2-8, 2-27
 - Insert Empty Line 2-8
 - Move to 2-8
 - Search 2-8
 - Select 2-8
 - Signal Set 2-8
- Signal names
 - Entering 3-3
 - Searching for 1-37 - 1-38
- Signal selection button 1-6
- Signal states
 - Toggle 3-7
- Signal transfer
 - Group 2-24
 - Individual 2-24
- Signal transition
 - Force 1-68
 - Measuring separation between 3-7
- Signal waveform editor 1-14
- Signal waveforms 1-13 - 1-16
 - Applying 1-23 - 1-24
 - Creating 1-39
 - Formula-based 1-42
 - Ready-made 1-15 - 1-16
 - Summary 1-24 - 1-25
- Signal, Bus B-5
- Signals
 - Add 2-8
 - Applying 1-39 - 1-42
 - Color code 2-55
 - Definition 4-2
 - Delete 2-8
 - Formula-based, creating 1-40 - 1-42
 - Locating 1-37 - 1-39
 - Logical state 3-5
 - Moving 2-29, 3-7
 - Names 5-2 - 5-3
 - Names, saving 1-74
 - Rearrange 3-4
 - Select 2-8

- Types B-6
- Unknown 2-43
- View Connections 2-30
- Signals Selection field 1-7, 1-11, 1-37, 2-27
- Signals statement 4-2
- Sim. Till End option 2-53
- Simulation
 - Advantages 1-12
 - Backup 1-45
 - Basics 1-1 - 1-4, 1-6 - 1-24, 1-26 - 1-27, 1-29, 1-31, 1-33
 - Clocks 1-70 - 1-71
 - Error Reports 2-62 - 2-63
 - Files 2-60 - 2-61
 - FPGA designs 1-78 - 1-79
 - Functional 1-26 - 1-29, 2-48 - 2-49, 2-51
 - Glitch 1-26, 1-29 - 1-31, 2-50
 - HEX code 2-22
 - Initialize 2-43
 - Introduction 1-1 - 1-3, 1-5 - 1-25
 - Large memory devices 1-62
 - Line (layout) delays 1-66 - 1-67
 - Long 1-61, 2-53
 - Netlist 1-71, 1-73 - 1-75, 1-77, 1-79
 - Override Outputs 2-51 - 2-52
 - Power On 2-46, 2-48
 - Precision 1-42 - 1-43, 2-9, 5-5
 - Reference points 1-58
 - Result 1-26 - 1-27, 1-29, 1-31, 1-33, 2-53
 - Run 2-42 - 2-43, 2-45, 2-47
 - Running long 1-60, 2-53
 - Running time 2-53
 - Save/Load Results 2-60 - 2-62
 - Selective 1-52 - 1-54, 2-51 - 2-52
 - Speed 1-52
 - status, current 1-59
 - Step 2-11, 2-42 - 2-43
 - Stop 2-9, 2-53
 - Switches, manipulating 1-65
 - System-level 1-79
 - Test vector limit 1-61
 - Time estimate 1-43 - 1-44, 2-9
 - Timing Mode 1-31 - 1-33, 2-50
 - Timing Mode, summary 2-51
 - Unit delay 2-49
 - Very large designs 1-52
 - Simulation results
 - Backup 1-44
 - Load 2-60 - 2-62
 - Print 2-71 - 2-73
 - Save 2-60 - 2-62
 - Simulation Stop option 1-65, 2-53
 - Simulator
 - Fault 2-11
 - Glitch 1-26
 - Main Toolbox 2-2
 - Main Window 2-1 - 2-4, 2-12, 2-30, 2-34, 2-74
 - Missing Models 2-15
 - Step window 2-2
 - Title bar 2-2
 - Simulator data
 - Locating 1-38
 - Simulator macros 5-1
 - Simulator window 2-1 - 2-4, 2-12, 2-30, 2-34, 2-74
 - Binary counter status 2-3
 - Long step setup 2-3, 2-43, 2-50
 - Long step simulation button 2-3
 - Main toolbox 2-2, 2-11, 2-50, 2-54 - 2-55
 - Maximize button 2-3

- Menu bar 2-3
 - Minimize button 2-2
 - Power On button 2-4
 - Schematic editor button 2-3
 - Search backward button 2-3
 - Search button 2-3
 - Search forward button 2-3
 - Short step setup 2-3, 2-50
 - Short step setup 2-42
 - Short step simulation button 2-3
 - Simulation mode setup 2-3
 - Simulation time display 2-3
 - Step window 2-2
 - Stop simulation button 2-4
 - System menu 2-2
 - Title bar 2-2
 - Snap option 3-6
 - Space B-2
 - States, logical
 - List 1-22 - 1-23
 - Status line 3-5
 - Steps
 - Long 2-42
 - Short 2-42
 - Stimulator
 - Add 2-8
 - Assigning 1-15 - 1-16
 - Binary counter 1-18
 - Clock 1-15 - 1-16
 - Definition 1-14
 - Formula 1-15, 1-19 - 1-21
 - Stimulator menu 2-8
 - Add Stimulators Mode 2-8
 - Chip Controlled 2-8
 - Connect 2-9
 - Delete 2-9
 - Delete All 2-9
 - Disconnect 2-9
 - Override Mode 2-8
 - Stimulator Selection window 1-15, 1-21, 2-38
 - Stimulus signals
 - Creating 1-12
 - Summary
 - Signal waveforms 1-24 - 1-25
 - Switch Setting window 1-65
 - Switch Settings 2-10, 2-67
 - Switch Settings option 1-65
 - Switch window
 - Minimized 2-68
 - Switches
 - Manipulate 1-65
 - Multiple 2-68
 - Position 2-68
 - Toggle 2-67
 - Switching between schematic & external netlist 1-75
 - Syntax specification 4-3
 - System-level
 - Simulation 1-79
 - System-level design tools 1-79
- T**
- Tab B-2
 - Tag button 1-64
 - Tag conditions 2-56 - 2-57, 2-59
 - Creating 1-63 - 1-64
 - Deleting 2-10
 - Locating 1-64 - 1-65, 2-56, 2-58
 - Search mode 2-58
 - Setting 2-10, 2-57
 - Signal transitions 2-57
 - Tag feature 2-56
 - Tag option 1-62, 2-55
 - Tag search mode 1-64
 - Technology Application Notes 1-34 - 1-70
 - Technology Change 2-65
 - Technology Selection window 1-33

- Terminal Type B-5 - B-6
 - Test Clock Rate 2-20
 - Test Editor 3-1
 - Create menu 3-3
 - Edit menu 3-4
 - File menu 3-2
 - Main menu 3-2
 - Options menu 3-7
 - View menu 3-5 - 3-6
 - Test PLD 2-8, 2-18 - 2-19
 - Test PLD window 2-19
 - Test points
 - Selecting 1-7 - 1-12
 - Selecting in hierarchical designs 1-9 - 1-10
 - Test vector 1-1 - 1-3, 1-8 - 1-9, 1-12 - 1-14, 1-16, 1-21 - 1-22, 1-25 - 1-26, 1-29 - 1-31, 1-39, 1-44 - 1-45, 1-58 - 1-60, 1-62, 1-70, 1-74, 1-76, 2-4, 2-6 - 2-8, 2-16, 2-18 - 2-20, 2-31 - 2-37, 2-39 - 2-41, 2-50, 2-52, 2-60 - 2-62, 2-69, 3-3, 3-6, 5-1, 5-6, 5-10, 5-29, 5-36 - 5-38, 6-1, 6-6 - 6-7, A-2, A-4
 - ASCII Files 2-40, 2-62
 - Binary Counter 2-34
 - Clocks 2-39
 - Design Custom 2-35 - 2-36
 - Edit 2-41
 - External Editor 3-1 - 3-7
 - File 2-61, 5-6 - 5-7
 - Formula 2-37
 - Keyboard 2-34
 - Load 2-7
 - Load ASCII 1-75, 2-7, 2-40
 - Moving to another design 1-75
 - Save 2-7
 - Save ASCII 1-75, 2-7
 - Set, define 2-8
 - Test vector limit
 - Overcoming 1-61
 - Test vector macro editor 1-14
 - Test Vector State Selection window 2-35 - 2-36
 - Time intervals
 - Measuring 2-55
 - Time relationship 2-55
 - Time units 5-4 - 5-5
 - Timing diagram
 - Print 2-71
 - Timing parameters 1-48, 2-64 - 2-65
 - Timing performance 1-32
 - Timing precision 5-5
 - Timing simulation 1-31 - 1-34, 2-50
 - Summary 2-51
 - Timing specification 2-64
 - Edit 2-10
 - Edit window 2-65
 - Timing Violations 1-31, 2-50, 2-62
 - Device-related 1-50 - 1-51
 - Toggle button 2-68
 - Toggling switches 2-67 - 2-68
 - Tracking errors through a design 1-75 - 1-76
 - Transport delay 2-10
- U**
- Uni-directional terminal B-6
 - Unit delay simulation 2-49
 - Unit propagation delays 2-50
 - Unknown signal conditions 2-43
 - Unknown signals 2-43
 - Utilities menu 2-10
 - Breakpoints 2-11, 2-71
 - Error Viewer 2-11, 2-63
 - Fault Simulator 2-11
 - Hierarchical Viewer 2-11
 - Internal State Editor 2-11
 - Macro 2-11
 - Memory Configurator 2-11
 - Memory Editor 2-11, 2-22, 2-59
 - Selective Preset 2-11, 2-44
 - Simulation Step 2-11
 - VHDL Debugger 2-11
 - Waveform Viewer 2-11
 - Utilities menu
 - Breakpoints 6-1
- V**
- Values
 - Change generic 2-10
 - VHDL debugger 2-11
 - View 2-11
 - Agents' registers E-4
 - Viewer
 - Error 2-11
 - Hierarchical 2-11
 - Views compatible macros 5-7 - 5-8, 5-10, 5-12, 5-14, 5-16, 5-18, 5-20, 5-22, 5-24, 5-26, 5-28
 - After 5-8 - 5-9

- Assign 5-9
- Breakpoint 5-9 - 5-10
- Breakpoint, event format 5-10
- Check 5-11 - 5-12
- Clock 5-12
- Comments 5-7
- Cycle 5-13
- Display 5-13 - 5-14
- Echo 5-14
- Every 5-14 - 5-15
- Execute 5-15
- Force high 5-16
- High 5-15
- Low 5-16
- Macro (command) loops 5-8
- Network 5-17
- Pattern 5-17 - 5-18
- Print 5-18 - 5-19
- Quit 5-19
- Radix 5-19
- Release 5-20
- Report 5-20
- Restart 5-20
- Restore 5-20
- Run 5-21
- Save 5-21
- Sim 5-21
- Stepsize 5-22
- Ticksize 5-22
- Time measurement 5-22
- Unknown 5-29
- Vector 5-22 - 5-23
- Watch 5-23
- Wfm Aperiodic 5-23 - 5-24
- Wfm Decrement 5-25
- Wfm Divide 5-25 - 5-26
- Wfm Increment 5-26
- Wfm Multiply 5-26 - 5-27
- Wfm Periodic 5-27
- Wfm Rotate Left 5-27
- Wfm Rotate Right 5-28
- Wfm Shift Left 5-28
- Wfm Shift Right 5-29
- Voltage, High A-3
- Voltage, Reference A-3
- W**
- Waveform
 - Analyze 2-54
 - Comments 2-9
 - Display 2-41
 - Formula 2-9
 - Insert Comments 2-58
 - Logical state 3-5
 - Markers 2-9
 - Measure Intervals 2-9, 2-55
 - Measurement mode 2-55
 - Print 2-71, 2-73
 - Scale resolution 2-42
 - Select blocks 3-2
 - Select Scale 2-41
 - Shape 1-40
 - Spacing section 2-72
 - Symbols and Colors 2-53, 2-55, 2-57, 2-59
 - TAG Search 2-56, 2-58
- Waveform Editor
 - Graphical 1-21 - 1-23, 2-9
- Waveform Menu 1-22, 2-9
 - Comments 2-9, 2-58 - 2-59
 - Edit 2-9, 2-36
 - Formula 2-9, 2-37 - 2-38
 - Markers 2-9
 - Measurements 2-9, 2-55
- Waveform Viewer 2-4, 2-6, 2-11, 2-58
- Waveform Viewer window 1-7, 1-9, 2-4 - 2-6, 2-24
- Waveforms window 2-7
 - Test vector entry 2-4
- Waveforms window areas 2-4 - 2-6
 - Blue cursor location 2-6
 - Bus On/Off 2-5
 - Current Logical State 2-7
 - Display Comments On/Off 2-5
 - I/O Attribute field 2-6
 - Logical States 2-6
 - Measurements On/Off 2-5
 - Red cursor location 2-6
 - Ruler On/Off 2-5
 - Scale Adjustment field 2-6
 - Scale Display 2-6
 - Screen resolution 2-6
 - Select Probes 2-6
 - Signal entry 2-4
 - Signal field 2-6
 - Signal waveform display 2-6
 - Stimulator field 2-7
 - Stimulus 2-6
 - Tag column 2-57

- Test vector entry 2-4
- Time Scale Display 2-6
- Waveform Delete 2-5
- Waveform window setups 2-4
- Zoom In 2-4, 2-53
- Zoom Out 2-4, 2-53
- Zoom, dynamic 2-53
- Waveforms window areas 2-4 - 2-6
 - Measurements On/Off 2-56
- What-if analysis 1-21, 1-40
- Window menu 2-11
- Worst-case conditions 2-65
 - Checking for 1-47 - 1-49
 - Hold time 1-47
 - Setup time 1-47, 1-49

Z

- Zoom
 - Dynamic 2-53