

October 1993

090-0511-001

Data I/O has made every attempt to ensure that the information in this document is accurate and complete. Data I/O assumes no liability for errors, or for any incidental, consequential, indirect or special damages, including, without limitation, loss of use, loss or alteration of data, delays, or lost profits or savings, arising from the use of this document or the product which it accompanies.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without written permission from Data I/O.

Data I/O Corporation
10525 Willows Road N.E., P.O. Box 97046
Redmond, Washington 98073-9746 USA
(206) 881-6444

Acknowledgments:

Data I/O is a registered trademark and Synario, Project Navigator, Retargeting Library, and ABEL-HDL are trademarks of Data I/O Corporation.

Data I/O Corporation acknowledges the trademarks of other organizations for their respective products or services mentioned in this document.

© 1993 Data I/O Corporation
All rights reserved

Table of Contents

Preface

Customer Support	ix
Technical Assistance	ix
Sending a Fax	ix
Using the BBS	x
Bulletin Board Service	x
Warranty Information	x
End User Registration and Address Change	xi
Where to Look for Information	xi
Conventions	xii

1. Welcome to Synario

What Is Synario?	1-1
Freedom in Design Entry	1-1
Freedom in Design Processing	1-1
Designing with Synario	1-2
Design Entry	1-2
Functional Simulation	1-2
Optimization and Device Fitting	1-3
PLD Design Flow	1-4
Timing Simulation	1-5
What's in Synario	1-5
Projects and Sources	1-5
Processes	1-6
Customizing Processes	1-7

Simulating Your Design	1-8
Synario and Windows	1-8
Flexible Design Entry	1-9
Selecting a Device	1-10
Synario File Structure	1-11
Learning More About Synario	1-12
Exploring Synario's Documentation	1-12
Exploring Example Designs	1-12
2. Building a Design	
Introduction to Projects and Sources	2-1
Opening and Saving Projects	2-2
Opening a Project	2-2
Creating a New Project	2-2
Saving a Project	2-3
Creating Schematics and Behavioral Modules	2-3
Adding and Removing Source	2-4
Building a Hierarchical Project	2-4
Understanding Hierarchy	2-4
Building a Top-level Behavioral Module	2-6
Building a Top-level Schematic	2-6
Test Fixture Sources	2-7
Building Other Project Sources	2-7
3. Editing Your Design	
Editing Behavioral Modules and Test Stimulus	3-1
Editing Schematics	3-2
Editing Other Sources	3-2
Converting JEDEC Files to ABEL-HDL	3-2
4. Processing Your Design	
Running Processes	4-1
Viewing Reports and Output Files	4-2

Processing Steps	4-3
Displaying Processes	4-3
Device-independent Processes	4-3
Device-specific Processes	4-3
Working with Processes	4-4
The Process Window	4-4
Changing How a Process Runs: Properties	4-5
Processing Problems	4-6
Error Messages	4-6
Debugging Problems	4-6
Lost Clusters	4-6
5. Design and Retargeting	
Schematic Design Techniques	5-1
VCC and GND Symbols	5-1
Net Names and Symbol Instance Names	5-1
Create Bus and Net Names that are Unique	5-2
Place I/O Pins On Top Level Only	5-2
Other Design Considerations	5-2
Don't Rely On Case-sensitivity	5-2
Keep Projects Separate	5-2
Use Text Documents	5-2
Test Fixture Include file	5-3
Designing for Device Independence	5-3
Horizontal and Vertical Retargeting	5-3
Device-independent Designs	5-4
Device-independent ABEL-HDL Modules	5-4
Device-independent Schematics	5-6
6. Working with Devices	
When You Need to Select a Device	6-1
Selecting a Device	6-1
Automatic Process Updating	6-2
What Happens When You Select a Device?	6-2

How Processes Change with a Device Change	6-2
How Properties Change with a Device Change	6-3
How Schematic Symbols Change with a Device Change	6-3
Device Kits	6-4
What Is a Device Kit?	6-4
How Do You Use a Device Kit?	6-4
Accessing Device Kit Help Files	6-5

Appendixes

A. Strategies and Properties

Properties	A-2
Advanced Properties	A-2
Default Properties	A-2
Setting Properties	A-3
Strategies	A-4
Creating New Strategies	A-6
Changing the Default Strategy	A-6
Deleting and Renaming Strategies	A-7
Saving Strategies	A-7

B. Synario Configuration

Synario Project Navigator	B-1
Sizing the Sources and Processes Window	B-1
Text Editor and Report Viewer	B-2
Schematic Editor, Symbol Editor, and Hierarchy Navigator	B-2
Synario Project Navigator Configurable Menu	B-3

C. The Synario INI Editor

Saving Changes	C-2
The BINARY.INI File	C-2
Custom INI Files	C-3
Using the Synario INI Editor	C-3

Controls Menu	C-3
System Controls	C-3
Display Controls	C-4
Symbol Controls	C-6
Graphic Options	C-7
Sheet Layout	C-8
Sheet Sizes	C-9
Wave Controls	C-10
Global Nets	C-10
Colors	C-12
Wave Colors	C-14
Tools Menu	C-15
Symbol Tools	C-16
Schematic Tools	C-16
Navigator Tools	C-17
Processes	C-17
Attributes Menu	C-17
Symbol, Pin, and Net Attributes	C-17
Example Attributes	C-21
Global Attributes	C-26
Search Paths Menu	C-27
Project, Model, and Symbol Libraries	C-27
Libraries and Directory Structures	C-28
Program Directories	C-28
User Directories	C-29
Library Directories	C-29
Miscellaneous Control Options	C-32

D. PLD JEDEC Simulation

How Synario Simulates JEDEC Files	D-1
The Simulator Model	D-1
JEDEC and .tmv Vectors	D-2
JEDEC Simulation Flow	D-2
Report and Trace Types and Break Points	D-4

Trace Type: Brief	D-5
Trace Type: Clock	D-5
Trace Type: Detailed	D-6
Report Type: None	D-7
Report Type: Tabular	D-8
Report Type: Pins	D-8
Report Type: Macro-cell	D-10
Report Type: Wave	D-12
Simulation and Designs with Buffered Outputs	D-12
Simulation and Unspecified Inputs	D-13
Simulation for Designs with Feedback	D-13
Register Preloads in the Simulator	D-17
Test Vectors and Simulation	D-17
Debugging State Machines	D-17
Multiple Test Vector Sections	D-18
Automatic Signal Selection	D-19
Don't Cares in Simulation	D-20
Preset and Preload Registers	D-23
Special Preset Considerations	D-23
TTL Preload	D-25
Supervoltage Preload	D-28
Preset/Reset Controlled by Product Term	D-31
Preset/Reset Controlled by Pin	D-32
Powerup States	D-32
Devices with Clock Inputs	D-33

Preface

The Preface includes details about obtaining technical assistance and warranty service. The Preface also explains the Bulletin Board Service, where to find information, and typographic conventions used in the manuals.

Customer Support

For technical assistance and warranty service, contact your local distributor. An updated list of Synario distributors is included in your Synario Design Entry package.

Technical Assistance

You can contact your Synario distributor for technical assistance by calling, sending a fax or electronic mail (e-mail). You can also access the Data I/O Bulletin Board Service (BBS) for product information.

To help us give you quick and accurate assistance, please provide the following information:

- ◆ Product version number
- ◆ Product serial number (if available)
- ◆ Detailed description of the problem you are experiencing
- ◆ Error messages (if any)
- ◆ Device manufacturer and part number (if device-related)

When you call, please be at your programmer or computer, have the product manual nearby, and be ready to provide the information listed above.

Sending a Fax

Fax the information listed above with your name, phone number, and address to your Synario distributor listed on the Synario Distributors sheet.

Using the BBS

To reach Data I/O via the BBS, include your name, phone number, e-mail address, and the information listed above in a message, and send it to the BBS as described in the following section.

Bulletin Board Service

From the Data I/O Bulletin Board Service (BBS) you can obtain a wide range of information on Data I/O products, including current product descriptions, new revision information, technical support information, application notes, and other miscellaneous information.

Using the BBS, you can access device support information, request support for a particular device, and leave messages for the BBS system operator or other customers. The BBS also includes many downloadable DOS utilities.

Multiple lines are available, all supporting 1200/2400/9600/19200 baud, with U.S. Robotics Dual/HST V.32bis/V.42bis modems. The modems are set to 8 data bits, 1 stop bit, and no parity. Online help files provide more information about the BBS and its capabilities.

BBS numbers for all countries are as follows:

Country	BBS Number
France	+33-(0)13-9562699
Germany	+49-(0)89-8585833
Norway	+47-(0)66-780445
Sweden	+46-(0)8-7391037
United Kingdom	+44-(0)734-448862
United States	+1-206-882-1976 +1-206-861-6959

Warranty Information

Data I/O Corporation warrants this product against defects in materials and workmanship at the time of delivery and thereafter for a period of ninety (90) days.

The foregoing warranty and the manufacturers' warranties, if any, are in lieu of all other warranties, expressed, implied or arising under law, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Data I/O maintains customer service offices throughout the world, each staffed with factory-trained technicians to provide prompt, quality service. For warranty service, contact Data I/O Customer Support at the numbers listed at the front of the Preface.

End User Registration and Address Change

If the end user for this product or your address has changed since you registered the product, please notify your Synario distributor. This ensures that you receive information about product enhancements. Be sure to include the product serial number.

Where to Look for Information

If you want to learn about	Look here
Installation	<i>Getting Started</i>
Menu operation	Online Help
Quick Start	Synario Online Tutorial Online Help (F1)
Design tutorials	<i>Getting Started</i>
Device-specific information	Device Kit manual Device Kit Help
ABEL-HDL syntax	Online Help <i>Synario ABEL-HDL Reference</i>
Schematic commands	Online Help <i>Synario Schematic Editor Reference</i>
Simulation	<i>Synario Simulator User Manual</i>
Complex PLD designs	PLD Device Kit <i>Getting Started</i>
FPGA designs	FPGA device kit <i>Getting Started</i>

Conventions

The conventions used in this manual are described below:



This symbol indicates that the information is available in Synario's online help system.

Keys and Key Combinations

An instruction for pressing two keys at once, such as ^Z (Control and Z), is represented by two keys separated by a plus, such as Ctrl + Z

A key combination like Esc, Ctrl + T means press and release Esc, then press Ctrl and T at the same time.

Variable Input

Variable input is italicized and should be replaced with the requested information. For example, "enter copy *filename.hex*" means type "copy" just as you see it and replace *filename.hex* with the name of your file.

Optional Input

Optional items of a command are shown in brackets.

[option1] [option2]...[optionn]

Items separated by a vertical bar (for example, OR | OR | ...) are mutually exclusive; that is, only one of the options listed can be specified.

Displayed Text

Text displayed on the screen appears in a typewriter-like typeface.

You will see this text displayed on the screen.

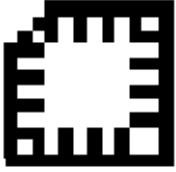
Menu Items, Processes and Properties

Menu items are entered with the menu name, a colon, and the menu selection. For example, File: Exit means to select Exit from the File menu.

Processes and properties are entered as they appear on the screen.

Keywords

Keywords and filenames appear in quotations or bold text.



Chapter 1

Welcome to Synario

The Synario Universal FPGA Design System brings together several FPGA design tools into a single Windows application. Synario streamlines the device design process, freeing you to concentrate on the details of your design. This chapter explains what Synario is and how you approach device design with Synario.

If you'd like to start using Synario right away, *Getting Started with Synario* gives you quick information on installing and starting Synario, along with tutorials to get you started designing in Synario.

What Is Synario?

Freedom in Design Entry

Synario is a design entry system for FPGAs. Synario helps you create designs that are

- ◆ Composed of either schematics or HDL modules, so you can use the best features of both design entry methods.
- ◆ Hierarchical, so you can create designs from the bottom up or from the top down.
- ◆ Device-independent, so you can easily retarget your designs using Synario's generic behavioral language and retargetable schematic symbols.

Freedom in Design Processing

Synario also gives you freedom in processing your design. You can

- ◆ Design with or without selecting a device, and change the device at any time. Synario updates the processes to reflect the requirements of the new device.
- ◆ Develop and reuse different strategies for processing designs.
- ◆ Simulate both the function and timing of your design.

- ◆ Simulate your design before choosing a device, and after implementing in a specific device.
- ◆ Run any process, with Synario automatically running prerequisite processing.

Designing with Synario

Synario consists of many parts, all of which are accessed through the Synario Project Navigator. The Project Navigator helps you keep track of all of the parts of your design, and keeps track of the processing steps necessary to move the design from the conceptual stage through to implementation in an actual device.

Design Entry

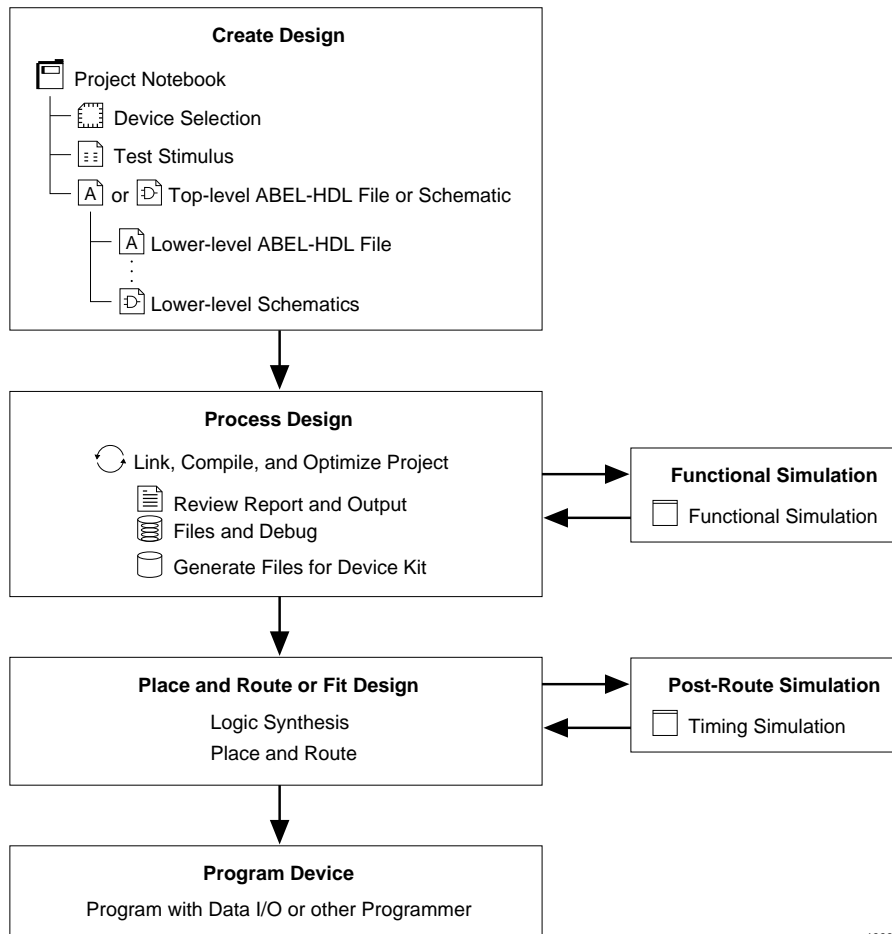
Synario supports two primary design entry methods; schematic and HDL. You can enter your design using either of these entry methods, or mix schematic and behavior entry in a single design. (Note: if your design is intended for a PLD, rather than for an FPGA, you cannot use schematics to describe the function of your design.)

In addition to the schematics and HDL portions of your design, Synario allows you to enter and keep track of other files not directly associated with the logic of your design. These files could include things like word processor documents (for your project specification), simulation test fixtures, or graphic state diagrams. These accessory files are kept for you in the Synario Project Notebook. Virtually any type of file can be imported into Synario and kept in the Project Notebook.

Functional Simulation

Functional simulation is a way to verify that your design functions as intended before attempting to fit it into a device. The Synario Simulator is a full-featured simulator that accepts Verilog test stimulus and can display the simulation results in a variety of formats, including waveforms. For designs that include a top-level schematic, Synario also allows you to view simulation results right on the schematic, with actual circuit values displayed for each design input and output. This feature is called cross-probing.

Figure 1-1
Synario Design Flow



1993-1

Optimization and Device Fitting

Synario's optional device kits allow you to target your design to a wide range of programmable devices. The devices supported by Synario Device Kits range from simple PAL and PROM devices to complex PLDs and FPGAs.

The optimization and fitting steps that Synario performs for you are determined by the type of device that you choose for implementation. There are two broad categories of devices, however, that can be described in terms of design flows. These categories are FPGAs and PLDs.

FPGA Design Flow

FPGA and PLD device architectures require different design processing. An FPGA-type device differs from a PLD primarily in its need for a place-and-route step. Place-and-route software is typically provided by the FPGA vendors, so the processing steps performed in the Synario environment include the use of tools that may not be integrated directly into the Synario Project Navigator.

The FPGA design flow also differs from the PLD design flow in the types of entry formats supported. Because FPGA device fitters (and place-and-route software) operate on netlist format data, you can use both schematics and behavioral (HDL) design entry formats. When you enter a design that consists of a combination of schematics and HDL components, Synario converts all of the input forms to a netlist representation before the design is moved to the device-fitting phase. For most FPGA type devices, the individual project sources (the schematics and HDL files) are combined (merged) during device fitting.

PLD Design Flow

The design flow for PLDs varies depending on the type of PLD chosen. For most types of PLDs, the design flow includes compilation of ABEL-HDL sources into the OPEN-ABEL format, and a linking step for designs that are composed of multiple source files. Some differences from FPGA design are that schematic sources are not supported for PLDs, and that multiple source files are combined (by linking) prior to the device-fitting process.

The PLD design flow also includes the capability for another form of simulation that is specific to PLDs: test vectors. Test vectors are commonly used to test the function of a PLD after it has been programmed, and the Synario JEDEC simulator can be used to simulate the program-and-test process. JEDEC simulation is supported in Synario for any device that is programmed using a JEDEC format programmer download file. The only PLD devices that are not currently support for JEDEC simulation are PROM devices (which do not use the JEDEC file format) and some of the larger complex PLDs. Refer to your PLD device kit documentation for complete information.

See Also

Appendix D, "PLD JEDEC Simulation"

Timing Simulation

After your design is successfully implemented in a programmable device, you can verify the timing performance of the resulting circuit. Timing simulation allows you to verify that your design will operate correctly at the speeds that you will be using in your target system.

To accurately simulate your design with timing, Synario builds a timing model of the programmed device using timing data. Once you have this simulation model, you can use the Synario Simulator to check the operation of the design and for timing-related circuit problems.

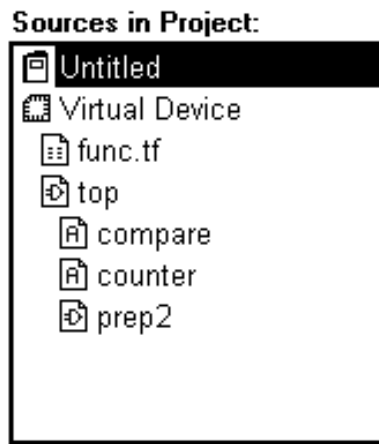
What's in Synario

Projects and Sources

Synario organizes a design by collecting all of the files into the project notebook. The Project Notebook lists the schematics and behavioral sources that create the logic of your design, testing files, and the device specification. The Project Notebook can also include any other design documents you want to keep with the design, such as design specifications, meeting notes or other supplementary files. Synario calls the notebook, documentation, and design pieces the *sources* of the project.

Each project is stored in its own directory to simplify archiving.

Figure 1-2
Sources in Project Window

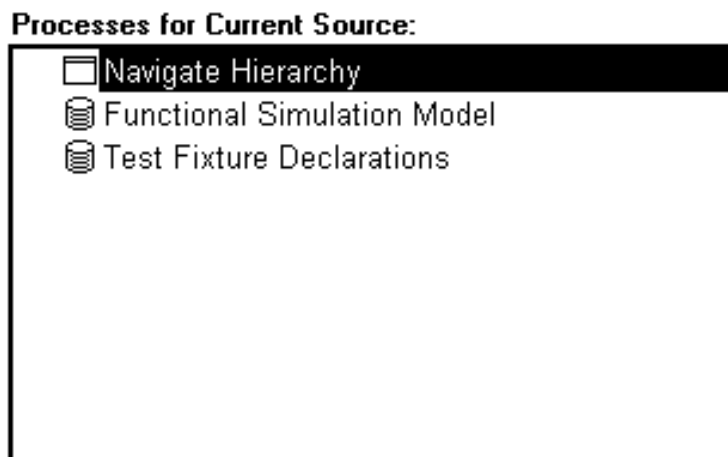


Processes

A Synario project may comprise many pieces, each of which needs to be handled in a different way. You might have schematics that must be drawn with a schematic capture program, tested, and then translated into netlists or other formats. You might also have behavioral modules that need to be compiled, optimized, tested, and translated into another format, and test fixtures that are used without processing. How each piece of a Synario project is processed is also dependent on the device selected.

All of the steps required to take your design from specification into a device are called “processes” in Synario, and Synario remembers them all for you. When you select a source or change the target device, Synario automatically displays the steps to get the job done, freeing you to concentrate on your design. All you have to do is double-click on a process to run it. Synario’s Auto-make capability automatically runs any prerequisite processes before running the process you select.

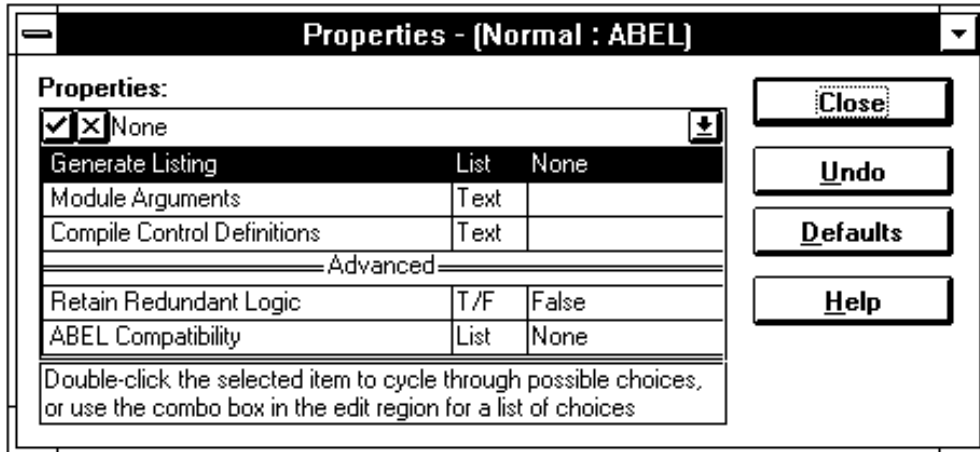
Figure 1-3
Processes for Current Source Window



Customizing Processes

Although Synario processes are usually already optimized for the device and source selected, there may be times when you want to change the way a source is processed. You can do this by changing the options, called *properties*, of a process in the Properties dialog box. You can save multiple sets of processing properties, each into their own *strategy*. For more information on Strategies and Properties, see Appendix A.

Figure 1-4
Properties Dialog Box



See Also



Specific processes and properties for each source and device are available in Synario's online help system.

Simulating Your Design

The Synario Simulator is an option that provides functional simulation for all designs, and timing simulation for devices that support simulation. Simulation is a process for a test stimulus source.

Designs targeted to most types of PLDs use ABEL-HDL test vectors to simulate the JEDEC programmer load file.

See Also

If you have the Synario Simulator, see the *Synario Simulator User Manual* for information on how to simulate your design.

For simulation support information, see your device kit documentation.

Synario and Windows

Synario is a Windows program and uses standard Windows operations. If you are familiar with how a Windows application works, you already know a lot about how Synario works. Some of the ways you can use your Windows experience in Synario are described below:

Drag and Drop from the File Manager

You can drag and drop any file from the Windows' File Manager into a Synario project as a project source. Any source that Synario does not recognize as part of a device design is stored under the Project Notebook as a document. This function is equivalent to choosing Import from the Source menu in the Synario Project Navigator.

Double-click Sources to Edit

The files in a Synario project are shown in the Sources in Project window. The Sources window displays the Notebook name, device selected, and the other sources of the project. Double-click on the device selection to select a new device. Double-click on any source to run the appropriate program with the specified file loaded — the same way you can in the File Manager.

Note *Synario sets up associations for .syn, .abl, .tf, and .sch during installation. You may need to set up file extension associations for other files. For instructions, see your Windows documentation.*

Note *You may also need to enable Use File Associations from the Options: Environment dialog box.*

Double-click Processes to Run Them

The steps to process the source highlighted in the Sources window are shown in the Processes for Current Source window. Double-click on a process in the Processes window to run that process. Synario automatically runs any prerequisite processes to complete the process you select.

Double-click on Reports to View Them

The reports generated by Synario processes are also listed in the Process window. You can double-click to view them, or select them and click the View button.

Double-click in the File Manager to Start Synario

Double-click on a Synario project file (*.syn) in the File Manager to start Synario with that project loaded.

Press F1 for Help

Synario comes with an extensive online help system. Press F1 to access context-sensitive help.

Flexible Design Entry

To take advantage of Synario's ability to change devices, enter your design so that it does not require a particular device; this type of design is called device-independent. Synario's Schematic Editor and HDL language both have device-independent features:

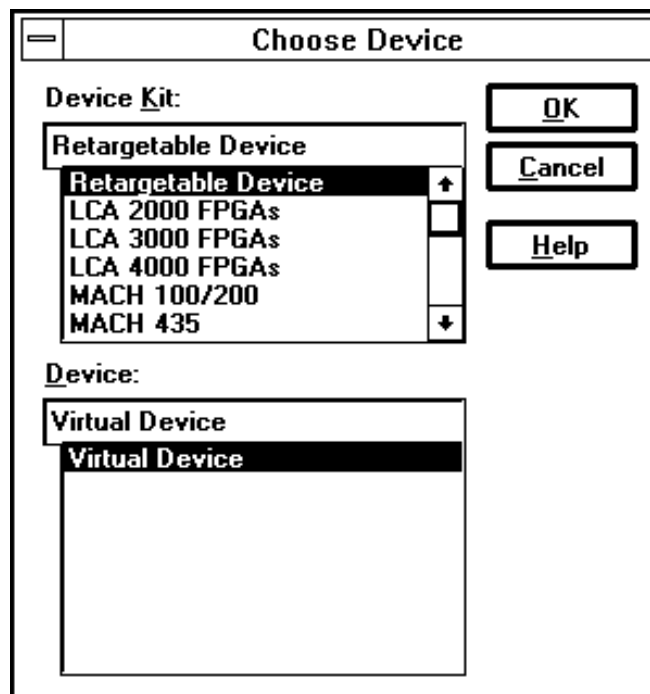
Design Type	Device-independent Features
Schematic	Use the Synario Retargeting Library to enter your schematics. These symbols are supported by all FPGAs and are mapped to the device-specific symbols before place-and-route. (Note: PLDs do not support schematic entry.) See the <i>Synario Schematic Editor Reference</i> for more information.
HDL	Use ABEL-HDL's device-independent language features. See the <i>Synario ABEL-HDL Reference</i> for more information.

Selecting a Device

When you do want to select a device, Synario allows you to become increasingly more specific about the device you want to use. You will need to select (target) a device to do device-specific design processing, such as logic synthesis and place-and-route.

Device-specific functions are provided through device kits. You should have received one or more device kits with your Synario package. Synario device kits are available for FPGAs, such as LCAs and MAX devices. There are also device kits for PLDs and complex PLDs.

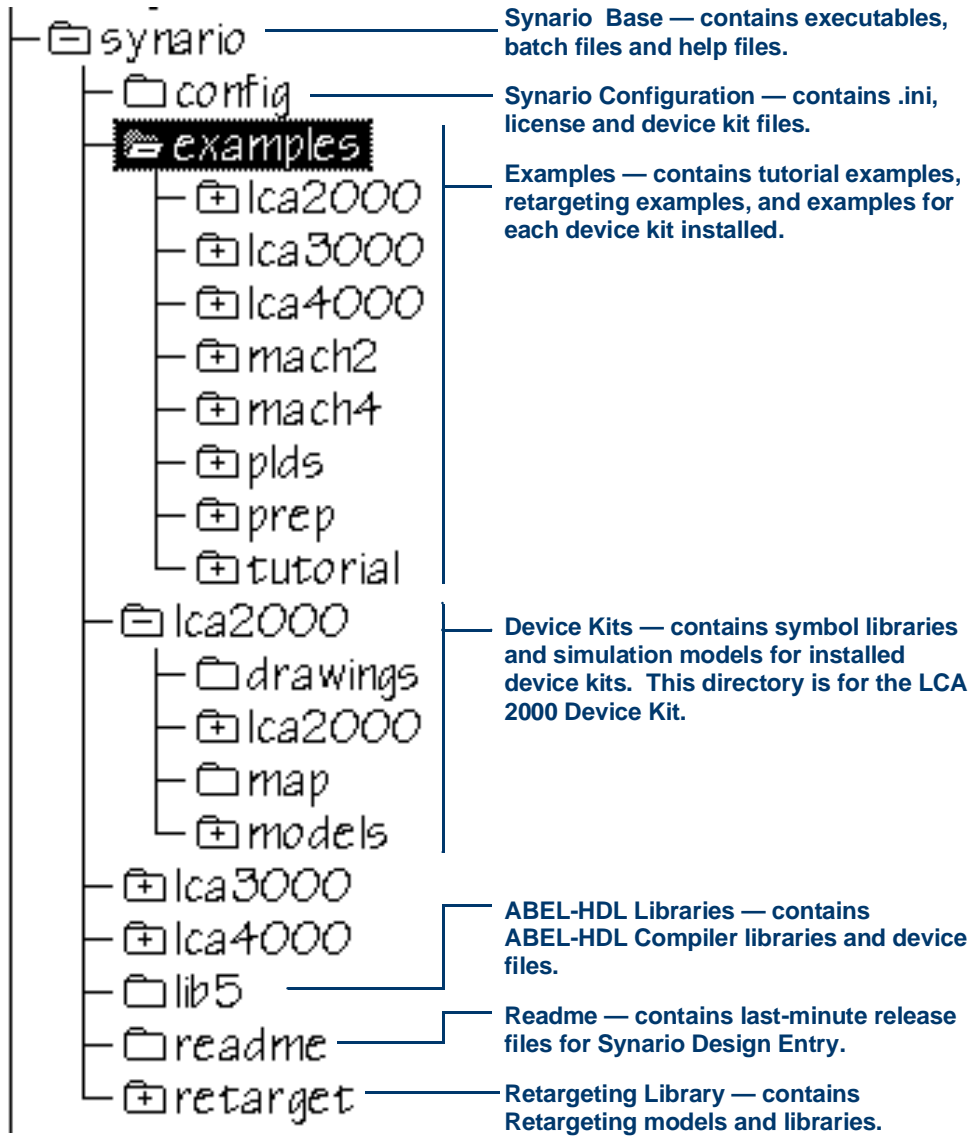
Figure 1-5
Selecting a Device



Synario File Structure

The files installed by Synario Design Entry are shown below.

Figure 1-6
Synario File Structure



Learning More About Synario

Exploring Synario's Documentation

The following table shows where to find things in Synario documentation. Most information is also available in Synario's online help system.

To Learn About	See
<i>Design Entry Topics</i>	
Device-independent designs	Chapter 5, "Design and Retargeting" <i>Synario ABEL-HDL Reference</i> <i>Synario Schematic Editor Reference</i>
Retargeting Library	<i>Synario Schematic Editor Reference</i>
<i>Design Building Topics</i>	
Creating, importing and editing sources	Chapter 3, "Building Your Design."
<i>Design Processing Topics</i>	
Selecting a Device	Chapter 6, "Working with Devices"
Processing your design	Chapter 4, "Processing Your Design"
Strategies and properties	Appendix A, "Strategies and Properties"

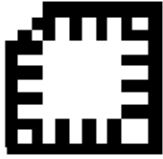
Exploring Example Designs

Synario comes with many example designs (called projects) that get you started creating your own designs. When you install a device kit, the installation program adds examples for the supported device.

To load an example design:

Select Open Example from the Synario Project Navigator File menu.

OR Double-click the *filename.syn* file in the Windows File Manager.



Chapter 2

Building a Design

This chapter introduces projects and sources, and gives instructions on how to build a design with them. For examples of building designs with Synario projects, see your *Getting Started* guide.

Introduction to Projects and Sources

In Synario, a design that will be targeted to a device is referred to as a project. A project contains sources that

- ◆ Identify the project and list included sources
- ◆ Define the design's logic
- ◆ Define the target device
- ◆ Provide test stimulus to test the design

Projects are built from the following types of sources. The type of source is indicated by the icon to the left of the source name in the Source Window.

Source	Icon	Example
Project notebook		hiermult.syn
Device family		lca.fdk
Behavioral logic description		hierabel.abl
Structural logic description		hierschm.sch
Test Fixtures		hierstim.tf
Undefined source		(n/a)

See Also



Projects

“Opening and Saving Projects”
“Adding Source”
“Adding Non-Synario Project Files”

Opening and Saving Projects


Opening a Project


To open an existing Synario project, select Open from the Synario File menu. Each Synario project is stored in its own directory. Example projects are installed in subdirectories under the Synario EXAMPLES directory.

- OR* Double-click on a Synario project file, *filename.syn* in the Windows File Manager.
- OR* Drag a Synario project file, *filename.syn* into Synario.

Creating a New Project

To create a new Synario project:


1. Select New from the File menu
2. Select a directory for the project, or select the Create Directory button and enter a name for a new directory. All of the files in your project will be stored in this directory.
3. Enter a filename for the project file.
4. Double-click on the project notebook icon  in the Source window, and enter a title for the project. The project title can be as long as you like; however, only the first 20 will show. The title can contain spaces and any other keyboard character except tabs and returns. The title is stored in the project file and is included in project output and report files created during processing.

Hint To rename the project, double-click the project icon  in the Source window.

Saving a Project

To save a project,

5. Select Save or Save As from the File menu. If you select Save As, Synario asks for a filename to save the project to.

OR Click on the Save button  in the toolbar.

What is Saved

Saving a project saves a project file (.syn extension) with the following information:

- ◆ The title of the project.
- ◆ The sources in the project
- ◆ The strategy associated with each source

Synario also tells the schematic and text editors to save when you save a project.

When you select Save As to save a project to another directory, Synario copies all of the project files to that directory.

Creating Schematics and Behavioral Modules

The design description (logic) for a project is contained within two types of source:

- ◆ Schematics
- ◆ Behavioral modules (for example, ABEL-HDL source)

One source file in a project is the top-level source for the design. The top-level source defines the inputs and outputs that will be mapped into the device, and references the logic descriptions contained in other, lower-level source. The referencing of another source is called “instantiation.” Lower-level source can also instantiate source to build as many levels of logic as necessary to describe your design.

Note *If you build a project with a single source, that source is automatically the top-level source.*

See Also

“Building a Hierarchical Project”
" Selecting a Device"

Adding and Removing Source

To add source to a project:

Select New from the Source menu, or click on the New button underneath the Sources in Project window.

OR Select Import from the Source menu.

OR Drag and drop a file from the Windows File Manager.

The new source is entered into the Source window in alphabetical order for each level of hierarchy following the project notebook and device entries.

To remove source from a project:

1. Select (highlight) the source in the Source window.
2. Select Remove from the Source window.

Note *Removing a source from a project does not delete the underlying file.*

Building a Hierarchical Project

Understanding Hierarchy

Hierarchical FPGA and PLD design consists of a top-level source that contains functional blocks linked to create the overall design. The functional blocks referenced in the top-level source are placeholders for lower-level logic descriptions, which may in turn contain functional blocks that reference even lower-level logic descriptions. The referencing of a logic description is called “instantiation.”

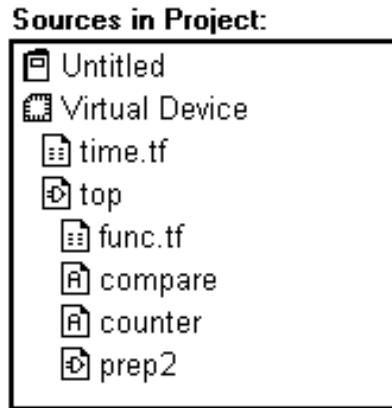
Both top-level and lower-level source can be either schematics or behavioral modules (in ABEL-HDL).

Note *A source can be referenced (“instantiated”) more than once. Also, a source can be both a lower-level and top-level source. For example, “compare” in Figure 2-1 could instantiate another file.*

Figure 2-1 shows what a hierarchical design looks like in the Synario Sources in Project list.

Figure 2-1

The Sources in Project Window



Untitled	The Project Notebook, which keeps track of all of the files that make up your project.
Virtual Device	The Device selected for the project (in this case, no device has been selected).
time.tf	A test fixture used to simulate the design implemented into the device.
top	The top-level source for the project, which instantiates the lower-level (indented) sources. In this case, it is a schematic.
func.tf	A test fixture used to simulate the function of the design, without a device.
compare, counter, prep2	Lower-level ABEL-HDL modules and schematic.

Note You cannot instantiate a top-level source from a source instantiated below that source. For example, you can't instantiate top from the compare source.

Building a Top-level Behavioral Module

A top-level behavioral module in ABEL-HDL uses the `Interface` and `Functional_block` keywords to instantiate lower-level files. The `Interface` keyword is also used in lower-level files to provide linking information to the upper-level module.

The ABEL-HDL file `pwmdac.abl` demonstrates the use of the `Functional_block` and `Interface` keywords in a top-level file. The file `counter.abl` demonstrates the use of the `Interface` keyword in a lower-level file.

These keywords are described completely in the *Synario ABEL-HDL Reference*.

See Also

“`Interface` (upper-level),” “`Interface` (lower-level),” and “`Functional_block`” in the *Synario ABEL-HDL Reference*

Building a Top-level Schematic

A top-level schematic (unless your design contains only one logic source) uses symbols to instantiate lower-level files. Select `Add: New Symbol` from the Schematic Editor menus to create a functional block symbol for a lower-level file.

If you are in a lower-level schematic, you can choose "This Block" in the "New Symbol" dialog box to automatically create a functional block symbol for the current schematic.

The Block Name is the name of the lower-level file, which can be another schematic or a behavioral module.

Schematic `top.sch` is a top-level schematic.

See Also

Synario Schematic Editor Reference

Test Fixture Sources

If you create or import a test fixture file, Synario prompts you whether to associate the test fixture with the device source or HDL/Schematic source.

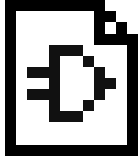
The project shown in Figure 2-1 includes two test fixtures (.tf extension). One associated with the device (time.tf) and one associated with the top-level source (func.tf).

You can double-click on a test fixture to edit it in the Text Editor.

Building Other Project Sources

You might have sources other than schematic and ABEL-HDL modules. These sources might include simulation test fixtures, documentation files, or other files related to Windows applications.

In most cases, you can drag and drop these files into your Synario project as needed. Any sources that are not part of the design logic description or simulation test fixtures are displayed under the Project Notebook.



Chapter 3

Editing Your Design

This chapter briefly explains the editors that come with Synario to help you start editing the sources that make up your logic design. More detailed information is provided in online help and the following manuals:

To Edit	Look Here
Behavioral Modules	<i>Synario ABEL-HDL Reference</i>
Schematics	<i>Synario Schematic Editor Reference</i>
Test Stimulus	<i>Synario Simulator User Manual</i> (if you have this option)

You can edit any of the sources that make up your project by double-clicking on them (if you have file associations set up in the Windows File Manager and have enabled "Use File Associations" in "Options: Environment."



In the editor for a source, you can press F1 or use the Help menus to access editor-specific help.

Editing Behavioral Modules and Test Stimulus

The Synario Text Editor provides several macros and templates to help you enter and edit behavioral modules written in ABEL-HDL, and test stimulus. You can edit an ABEL-HDL or test fixture source by double-clicking on it, or by selecting it in the source list and choosing Source: Open.

You can also use any ASCII editor to edit behavioral modules and test stimulus. You then import them into your Synario project using drag-and-drop from the Windows File Manager or using Source: Import.

See Also



"Menus" for information on Synario Text Editor menus and using templates.
Synario ABEL-HDL Language Reference for information on ABEL-HDL
Synario Simulator User Manual

Editing Schematics

Use the Synario Schematic Editor to edit schematic source. You can open the Schematic Editor on a schematic by double-clicking on the schematic name in the source list, or by selecting the source and choosing Source: Open.

See Also

The *Synario Schematic Editor Reference* for information on creating schematics and on the Retargeting Library.

Your device kit documentation for device-specific symbol libraries and generic symbol to device symbol mapping charts.

Editing Other Sources

Other sources can be edited by double-clicking if you associate their file extensions with appropriate editing applications in the Windows File Manager. See your application documentation for editing information.

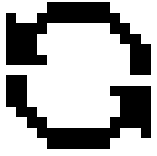
Note You can associate text files with the Synario Text Editor by choosing *File: Associate from the Windows File Manager*. See your Windows documentation for more information on file association in the File Manager.

Converting JEDEC Files to ABEL-HDL

If you have JEDEC files that you want to convert to ABEL-HDL files to use in a Synario project, you can use `jed2ahdl` in the Synario DOS window. You can bring up the Synario DOS window by pressing Ctrl+Alt+D.

The options for `jed2ahdl` are listed below:

```
jed2ahdl infile -o outfile -report mapfile
```



Chapter 4

Processing Your Design

This chapter explains how to process a project, including

- ◆ Running Processes
- ◆ Viewing Reports and Output Files
- ◆ Understanding Processing Steps
- ◆ Summary of Processes for Sources and Projects

See Also



Online help for Processes and Properties
Online tutorial

Chapter 2, “Building a Design”
Chapter 3, “Working with Sources”
Device kit documentation
Synario Simulator User Manual

Running Processes

You can run a process with one of the following procedures:

Use the Start or View Buttons:

1. Select the source you want to process. (To process the entire project, highlight the device.)
2. Select the process you want.
3. Click on the Start button to run the process; click on the View button to run the process and view the resulting file (for report and viewable output files).

OR **Double-click on a Process:**

4. Select the source you want to process. (To process the entire project, highlight the device.)

5. Double-click the end process you want. Synario automatically updates any intermediate steps necessary to complete the required process.

OR **Use the Process Menu**

1. Select the source you want to process. (To process the entire project, highlight the device.)
2. Select the process you want.
3. From the Process Menu (keyboard shortcut is Ctrl+P), select Start to run the process, or View to run the process and view the resulting file (for report and viewable output files). You can also select Force to run a process again that has already completed successfully.

See Also

“What Happens When You Select a Device?” in Chapter 6

Viewing Reports and Output Files

To view a report or output file:

1. Highlight the desired file.
2. Click on the View button or choose View from the Process menu.
3. When the process completes successfully, the File Viewer displays the file.

Processing Steps

Displaying Processes

Processes are associated with sources, so the processes displayed are only those processes for the highlighted source. Table 4-1 shows which source some common processes are associated with.

Table 4-1

Source Associations for Processes

Type of Process	Source to Highlight
Compiling	Behavioral
Simulation	Test Fixtures
Waveform Viewer	Schematic

Device-independent Processes

- ◆ Schematic processing and translation
- ◆ Behavioral processing and translation
- ◆ Hierarchy linking
- ◆ Design optimization
- ◆ Functional simulation — Functional Simulation can be done both before and after the design is routed into a device. See the *Synario Simulator User Manual* for more information.

Device-specific Processes

Device-specific processes change depending on the device selected.

- ◆ Synthesis — For some devices, logic synthesis might be performed. See your device kit documentation for details.
- ◆ Place and Route — Most device kits automatically assign pins and produce the appropriate output files for programming the device. See your device kit documentation for details.
- ◆ Functional and Timing Simulation — Because synthesis and place-and-route functions can significantly change your design, you might want to simulate it both before and after the design is routed. See the *Synario Simulator User Manual* for details.

Note *The Synario Simulator is an optional package.*

Note For most PLDs, you can do JEDEC simulation. See your PLD device kit, and Appendix D for more information.

Working with Processes

This section contains the following topics:






- ◆ The Process Window
- ◆ Changing How Processes Run — Properties

The Process Window

Each step required to process a project and its source is referred to as a Process. The processes available for the project and each type of source are different and vary based on the device selected. When you select (highlight) the project name or one of the sources in the Source Window, the processes available are displayed in the Process Window.




Types of Processes

The types of Processes available are shown below.

Icon	Process Type
	Run batch program
	Run Windows program
	Create Report File and View it
	Create Viewable Output File
	Create Non-viewable Output File

To see the status of a Process:

Look to the left of the process type icon for the process status icons:

Icon	Means process completed...
(No icon)	(Not started)
	Successfully, without errors or warnings
	With warnings (subsequent processes can continue)
	With fatal errors (processing halts completely)

Changing How a Process Runs: Properties

You can change the way a process runs by changing the options, called properties of the process.

To change the properties for a process:

1. Select (highlight) the process in the Process window.
2. Click on the Properties button. The Synario Property Editor displays the available properties for the selected process.
3. Double-click on a property to edit it. Changes take effect immediately and are saved to the current strategy.

To change properties for a different process or source:

Without closing the Property Editor, select the desired source and process. The Property Editor automatically updates the properties.

See Also

Appendix A, “Strategies and Properties” for more information.

Processing Problems

Error Messages



Error messages are available from online help by selecting Messages from the Problem Solving category on the help map. Device Kit error messages are available in the help file shipped with the device kit.

Debugging Problems

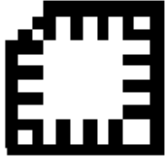
If you are having problems with Synario programs running under syndos, you can debug them by running a DOS window with the syndos.pif file. To access this window, press Ctrl+Alt+D. This gives you a DOS window with the same environment. Type `exit` to quit.

Lost Clusters

If you are having unexplained failures, run

```
chkdsk /f
```

to remove lost clusters from your hard drive.



Chapter 5

Design and Retargeting

This chapter covers project design techniques and designing for device independence. This information on conventions for designing with Synario will help you produce designs that are

- ◆ Easier to debug
- ◆ Retargetable to other device families
- ◆ Compatible with Synario designs created by other Synario users

Schematic Design Techniques

VCC and GND Symbols

The Synario Schematic Editor does not have VCC or GND symbols. Instead, you create connections to ground and VCC by naming the net "VCC" or "GND." Nets with those names are tied to the corresponding global signal. By creating a vertical net with the name "VCC" attached at the top, the VCC bar symbol appears. Likewise, by creating a vertical net with the name "GND" attached at the bottom, the ground symbol appears.

Net Names and Symbol Instance Names

During simulation, you may want to probe the design for certain nets or symbols. To improve post-synthesis debugging, you should name all of your intermediate nets with unique names, rather than having Synario provide automatic net names (which can be cryptic and difficult to sort out).

Create Bus and Net Names that are Unique

Bus names and net names, if identical, will "short out" in Verilog simulation because of the way Verilog breaks out bus signals. For example:

```
DATA[ 7:0 ]
```

and

```
DATA
```

will cross-connect during simulation, and cause strange and unpredictable errors. Avoid using buses and nets names which are identical.

Place I/O Pins On Top Level Only

I/O symbols, and thus pin assignments, should be kept in the top level source only. This makes designs much easier to simulate and understand later, as all of the device I/O can be found in one place, the top level source.

Other Design Considerations

Don't Rely On Case-sensitivity

Names should not rely on case to provide uniqueness. For example, don't use "FRED" and "fred" in the same design. Some tools in the Synario environment are inherently case-sensitive, though many are not case-sensitive. Duplicate names with different cases may internally "short" together during some back-end process, resulting in an invalid programming file, and/or incorrect simulation results. Some processes also modify the case of signals and names.

Keep Projects Separate

Keep projects separate in their own directories to make them easier to archive, easier to move, and easier for other engineers to work on.

Use Text Documents

Place design notes in separate text files associated with your project rather than documenting directly in the schematics and ABEL-HDL modules to make the logic easier to follow.

Test Fixture Include file

Use a Verilog `include` statement to include the automatically-generated test fixture declarations (.tffi) in your test fixture (.tf) files to minimize re-working the test fixture files.

Designing for Device Independence

This section describes how to create device-independent designs that can be easily retargeted to different devices. Designing for device-independence has many advantages. Device-independent designs mean you can:

- ◆ Take advantage of new devices as they become available with minimal maintenance.
- ◆ Learn one behavioral language and one symbol library for all devices, instead of learning a new one for each device you use.
- ◆ Try a design in several architectures to determine the ones in which your design fits best or runs fastest.
- ◆ Not be committed to the architecture you start your design with.
- ◆ Re-use your Synario designs in whole or in part.
- ◆ Draw your schematic without committing to a device.
- ◆ Perform functional simulation before device-specific features are specified.

Horizontal and Vertical Retargeting

There are two types of device retargeting: "horizontal" and "vertical."

Horizontal retargeting	Mapping a design created for one device into a similar device.
Vertical retargeting	Mapping several smaller designs targeted for small devices, linked together into a larger device.

Device-independent Designs

Device-independent ABEL-HDL Modules

Most of the ABEL-HDL language is device independent (for example, equations, truth tables, state machine descriptions, and test vectors). However, the ABEL-HDL compiler needs some information about how signals function in logic descriptions.

Information needed	Provided in ABEL-HDL with
Signal specifications	Istype attributes in the signal declarations and dot extensions on signals in the logic description
Special functions	Property Statements

Most common signal functions can still be described independent of a device with the pin-to-pin signal declarations. Some special features available in only certain classes of devices require device-specific (“detailed”) signal declarations.

Conditional Symbols

Some devices also have special functions that can be accessed through conditionally-compiled syntax. Each device kit has one or more special compile-time symbols that can be used to create conditionally-compiled designs. The table below shows a sample of the conditional symbols for various device kits. Refer to your device kit for more information on the conditional symbols supported.

Device Kit	Conditional Symbols
LCA2000	_LCA2000_, _LCA_
LCA3000	_LCA3000_, _LCA_
LCA4000	_LCA4000_, _LCA_
PLDs	_PLD_
MAX5000	_MAXPLUS2_, _MAX5000_
MAX7000	_MAXPLUS2_, _MAX7000_
MACH2	_MACH2_, _MACH_
MACH4	_MACH4_, _MACH_

An example of how to use conditional symbols in your ABEL-HDL source file is shown below:

```
@IFDEF _LCA_  
{  
  <LCA macros>  
}  
  
@IFDEF _PLD_  
{  
  <PLD macros>  
}
```

Pin-to-Pin Signal Specifications

Using ABEL-HDL's pin-to-pin syntax allows you to create logic descriptions that can be retargeted to different devices. Use the retargeting syntax whenever you have a choice to allow the module to be retargeted with little or no modification.

Detailed Signal Specifications

Detailed signal specification provide you with access to unique features found only in certain device families.

Note *You can combine pin-to-pin and detailed signal declarations. In most cases, the ABEL-HDL compiler can reconcile the requirements of both types. In some cases, however, the circuit function may be ambiguous and the compiler displays an error message.*

Syntax descriptions indicate whether the syntax is pin-to-pin or detailed.

See Also

“Pin-to-pin Vs. Detailed Module Descriptions” in the *Synario ABEL-HDL Language Reference*

Your device kit for device-specific ABEL-HDL property statements and dot extension restrictions

Device-independent Schematics

Synario Retargeting Library

Synario's Retargeting Library gives you device-independent flexibility for your schematics. If you use device-specific symbols in a schematic, you may have to redraw the schematic if you later retarget your design to a different device.

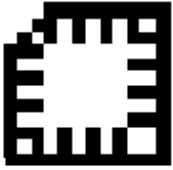
Device Kit Symbol Libraries

The symbol libraries that come with each device kit are device-specific. If you plan to stay in one device family, these libraries are all you need. However, since device kit symbols are tied tightly to the device they describe, retargeting a project with a device-specific schematic may require re-drawing the schematic using the new target device's symbol library. For this reason, we recommend the Synario Retargeting Library if you plan to use more than one type of device.

See Also

“Device-independent ABEL-HDL Modules”

Your device kit for device symbol libraries and Synario Retargeting Library mapping



Chapter 6

Working with Devices

There are many ways you can make it easier to take an existing design and change the device it maps into. These issues are discussed in the “Designing for Device Independence” section in Chapter 5.

When You Need to Select a Device

With Synario, you do not have to select a device to begin processing a design. You do need to select a device before you place and route your design or perform timing simulation (if you have the Synario Simulator).

You can, of course, specify a device at earlier stages in the design process.

Selecting a Device

To choose any device, you must have a Synario Device Kit and the device support for the device installed.

You select the device for your project in the Choose Device dialog box. You access it by double-clicking on the Device icon in the Sources window, or by selecting the Device icon and choosing Open from the Source menu.

- | | |
|------------|---|
| Device Kit | Lists the titles of the device kits currently licensed for this Synario installation. You pick a device kit from this list, and the Device box below is updated with the devices supported by that kit. |
| Device | Lists the supported device names licensed by the device kit. You can pick a device from this list, or enter a device name. If you enter a name, it must match a device in the listing (case doesn't have to match). |

Note *If the current project has sources that are not supported by the selected device, you will get an error dialog box. Either select a different device or remove the unsupported sources from the project. (For example, standard PLDs do not support schematics, so selecting a P22V10 for a project that includes a schematic will cause this to happen.)*

If your device change involves changing the underlying design environment (which is done automatically), Synario asks you to confirm the change.

Automatic Process Updating

You do not have to remember how to process your designs for different devices: when you specify or change the device for your design, Synario re-configures the processing steps automatically to reflect the processes required for the selected device.

What Happens When You Select a Device?

Changing the device to a device in a different device kits directly affects the Synario design environment. Some of these effects are immediately visible, while others are more subtle, but very important.

The following are affected when changing device kits:

Device	The device specified in the Source window changes.
Processes	The processes displayed in the Processes for Current Source window for all sources changes.
Properties	The properties available for each Process change.
Schematic symbols	The available symbol libraries change.

More detailed information about what is changed is given in the following sections.

How Processes Change with a Device Change

When you change to a device in a different device kit, the processing for the project changes. These changes are reflected in the following changes:

- ◆ The available processes change, especially device-specific process steps and netlists.

- ◆ The status of some processes that remain may return to “not processed” (that is, the green checks are removed and the process must be re-run for the new device).
- ◆ The properties available for processes.

How Properties Change with a Device Change

When you change to a device in a different device kit, the properties available for each Process change in the following ways:

- ◆ Properties are preserved for processes that did not change from the previous device kit, (like the ABEL-HDL process “Compile Logic”).
- ◆ Properties for new processes, and new properties for existing processes have default values assigned.
- ◆ For processes that are not available with the new device, the properties are no longer used or visible, but the settings are saved (if you switch back to the previous device kit, these properties will be the same).

How Schematic Symbols Change with a Device Change

When you change to a device in a different device kit, the schematic symbols that are available change. Each device kit supports the Synario Retargeting Library and the symbol library for the devices in the kit. This means that after changing kits, Add: Symbol in the schematic editor lists the available symbols for the device kit.

CAUTION

- ◆ You need to close and re-open the Schematic Editor and Hierarchy Navigator for the symbol availability changes to take effect.
- ◆ Schematics drawn with symbols from only the Synario Retargeting Library work without modification.
- ◆ Schematics drawn with symbols from a device-specific symbol library that is no longer available will be missing those symbols (you will get “Missing Symbol File” messages).

Note *If you use the Synario Retargeting Library to draw all of your schematics, you'll rarely have to redo a schematic when you select a different device.*

Replacing Missing Symbols in a Retargeted Schematic

To modify a schematic that is missing symbols after you've changed the device kit being used, do one of the following:

- ◆ Edit the schematic and select a different symbol from the new device kit library or the Synario Retargeting Library.
- OR*
- ◆ Create a new symbol and possibly an underlying schematic to support the new device kit.

For example:

Assume your original schematic was targeted to the MAX/FLEX device kit, and you used a symbol from the MAX/FLEX library. You then switch to the LCA3000 device kit, and this symbol is not supported. You get an error message “Missing symbol file *name.sym*” when you try to edit the schematic.

You can create a local symbol called *name.sym*, then draw a schematic called *name.sch* using symbols from the LCA3000 symbol library or the Synario Retargeting Library to provide the functionality of the MAX/FLEX symbol.

Device Kits

What Is a Device Kit?

A device kit contains all of the tools you need to create designs for that device or family of devices. So if you purchase the LCA device kit, you receive tools and documentation to help you create designs for LCAs. Device kits can contain all or some of the following items, depending on the processing requirements for the devices supported:

- ◆ Device Synthesis applications
- ◆ Place-and-route applications
- ◆ Functional and Timing Simulation models (for simulation with the optional Synario Simulator)
- ◆ Device-specific Synario project examples
- ◆ Device-specific design help files
- ◆ Device-specific symbol libraries
- ◆ Device optimization and pin assignment software

How Do You Use a Device Kit?

To use your device kit, follow the steps below:

1. Install the device kit software.
2. In your Synario project, double-click on the device icon and choose a device supported by the device kit.

The installation installs device kit files that Synario uses to update the processes available to you.

Accessing Device Kit Help Files

You can access the separate help file installed with your device kit using one of the procedures below:

Choose the device kit help file from the Synario Project Navigator Help menu.

OR In Synario's main help file, click on the Device Kit button.



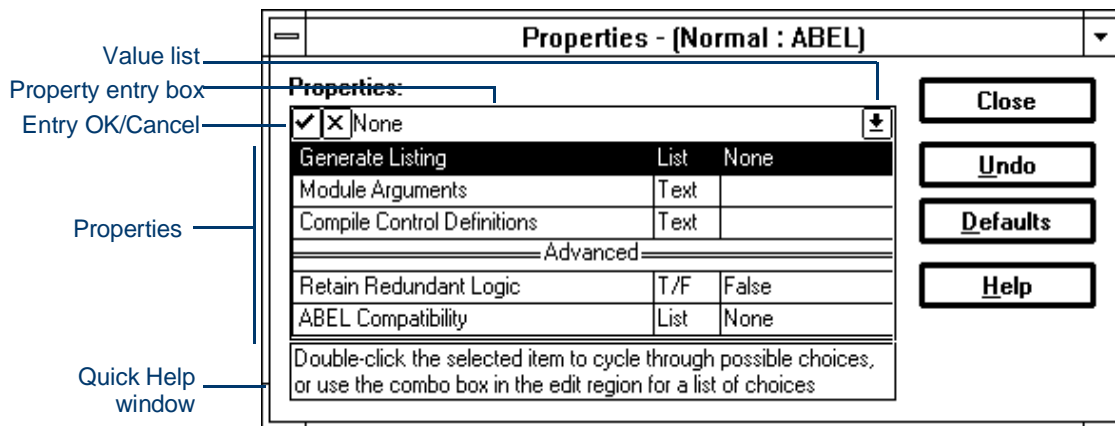
Appendix A

Strategies and Properties

Many processes have options, called properties, that tell Synario how you want the process carried out. You can modify these properties in the Properties dialog box.

To use the Properties dialog box, select a process in the Process list that you want to modify. If the process has properties, the Properties... button below the Process list will be active. Click on this button to open the Properties dialog box and edit the properties for the currently selected process.

Figure A-1
Properties Dialog Box



The title of the Properties dialog box window displays the name of the currently selected Strategy (Normal) and the type of source selected (Design). If you modify the properties shown above, you are modifying the properties in the Normal style, for Standard PLDs, for the currently-selected Design-level process.

Properties

There are three basic property types: True/False, Text, and List .

True/False True/False properties have "T/F" in the second column of the Properties dialog box. To change the value of a True/False property:

Double-click on the property.

OR Select the property and then press the value-list drop-down button to select True or False.

OR Press T or F when a True/False property is selected.

You can undo a change by pressing the Undo button.

Text Text properties take text strings defined by that property. To modify text properties

1. Select the text property in the property list. The current value of the text property (if any) is displayed in the property entry box (see Figure 1-4).
2. In the edit region, type in the text value for the property.
3. Click on the check (OK) button next to the edit region to accept the change. Click on the X (Cancel) button to cancel changes.

List List properties allow you to select one value from a list of possible choices. To cycle through all the possible values:

1. Select the List property.
2. Click on the value-list drop-down button and select one of the values.

OR Double-click on the List property. Each double-click causes the List property to change to the next possible value in the list.

Advanced Properties

Some properties in the Properties dialog box are advanced. Be sure to review the online help for these properties before changing them.

Default Properties

You can reset properties to the defaults for your device kit by clicking the Defaults button.

Setting Properties

To set properties:

1. Select a source (for example, select the device to change project properties, or select a logic source (schematic or ABEL-HDL module) to change source properties).
2. Select a process from the Process window.
3. Click on the Properties button, or choose Properties from the Process menu. The Properties dialog box displays the standard and device-specific properties for that process.
4. Click on the Advanced button for more properties.
5. Click on a property to select it.
6. Double-click on an option to toggle between choices. Click once to select an option and type in a new value, then select the green check to accept the change, or the red X to undo.

If you change your mind, click on the Undo button.

7. To change the properties for another process, click on the desired source and/or process in the Synario main window. (You may need to move the Properties dialog box.)

The properties for the selected process are immediately displayed in the Properties dialog box window.

Strategies

Strategies are advanced features of Synario that allow you to set up different processing options for different parts of a project. For example, you can use strategies to use one set of synthesis options for part of your project, and another set for the rest.

A strategy contains all of the property settings required to process the sources (such as an ABEL-HDL module) in the Sources in Project list, and a strategy can be assigned to one or more of the following types of source in a project:



Design



Schematics



ABEL-HDL modules

You can create multiple strategies and associate them with different sources in your design, allowing you to use different property settings for each source in your project.

When you associate a project source with a strategy, the source extracts its property settings from the strategy. For example, if you associate an ABEL-HDL source with a strategy named “Speed,” the ABEL-HDL module uses the ABEL-HDL process property settings in the strategy “Speed,” and ignores the property settings for the strategy “Normal.”

A Strategy is a collection of all the properties for all the processes. For example, you could set up a "Normal" strategy that has all of the ABEL-HDL Compile Logic and Reduce Logic properties set to optimize a module, and create a "No Optimize" strategy that has the Compile Logic and Reduce Logic properties set to not optimize the logic in a module (perhaps for timing purposes). You would then associate the "No Optimize" strategy with the sources that you do not wish to be optimized. Figure A-2 shows a graphical representation of a strategy.

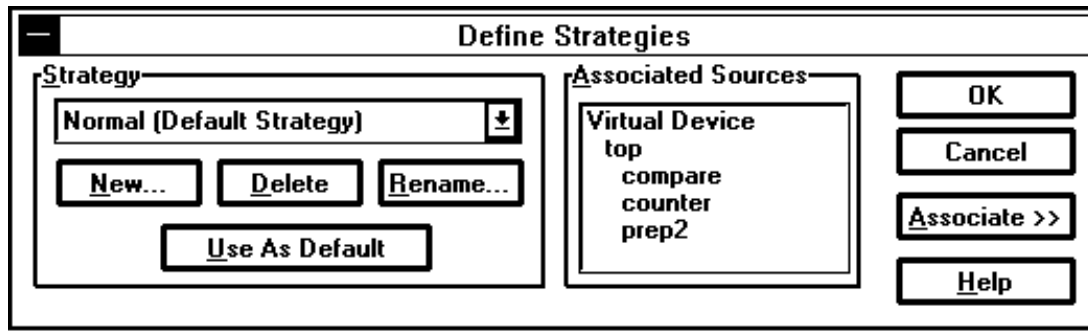
Figure A-2
Synario Strategies

Processes	Properties
<u>ABEL-HDL</u>	Generate Listing:None
Compile Logic	Retain Redundancy: False
Reduce Logic	Reduction Algorithm:By-Pin
XNF Netlist	Optimization Goal: Area
...	...
<u>Schematic</u>	...
EDIF Netlist	Compress EDIF:False
...	...
<u>Device</u>	...
Fit Design	Pin Preassignments:Keep
Create Fuse Map	Program Turbo Bits:False
Simulate JEDEC	Trace Type: Pins
Export to XACT	Back Annotation:XNFBA
...	...

Creating New Strategies

A project initially contains only the “Normal” strategy. You can create additional strategies and assign them to the different sources in your design. To create additional strategies, select Strategy from the Source menu to bring up the Define Strategies dialog box.

Figure A-3
Define Strategies Dialog Box



To create a new Strategy,

1. Click on the New button.
2. Type in a name for the new strategy.
3. Select an existing strategy to copy the new strategy’s properties from (or select System Defaults to get the default properties).
4. Click on OK (or press Enter).

Changing the Default Strategy

The “Normal” Strategy is automatically created when you start a new project in Synario. Normal is the default strategy and is assigned to all sources when they are added or imported into a project.

You can change the default strategy for a project to one of the strategies you have created.

To set a new default strategy:

1. Select the Strategy from the Source menu to bring up the Define Strategies dialog box.
2. Select the strategy that you want to be the new default from the Strategy list box
3. Click on the Use As Default button.

The strategy you selected will now be assigned to all new and imported sources in your project.

Deleting and Renaming Strategies

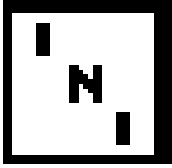
To delete or rename a strategy:

1. Select Strategy from the Source menu to bring up the Define Strategies dialog box.
2. Select a strategy.
3. Click on the Delete or Rename... button.
4. If you are renaming the strategy, enter a new name and click on the OK button.

Note *If you delete the Default strategy, the strategy at the top of the strategy list becomes the new Default Strategy. See “Changing the Default Strategy” if you want to change it.*

Saving Strategies

Strategies are automatically saved when you save your project.



Appendix B

Synario Configuration

You can change the Synario environment by changing settings in different configuration files. You access the configuration files through the Options menu in the program or through the Project Navigator Options menu.

Tool	How to modify	Configuration File(s)
Synario Project Navigator (synario.exe)	Options: Environment/Fonts	WINDIR\synario.ini
Synario Text Editor (synedit.exe)	Options menu	WINDIR\synedit.ini
Synario Text Viewer (synview.exe)	Options: Environment/Fonts	WINDIR\synview.ini
Schematic Tools (synario.exe)	Options: Schematic from synario.exe	\synario\config*.ini

Synario Project Navigator

You can set many environment variables in the Synario Project Navigator by choosing the Options menu, then Environment. The online help for the Environment dialog box discusses these options and the choices you have for changing them.

Sizing the Sources and Processes Window

You can change the size of the Sources in Project and Processes for Current Source windows, by setting it manually in synario.ini (which should be installed in your Windows directory).

To enlarge or shrink the Sources window:

Add the following line to synario.ini in the section [Data I/O Synario]

```
SplitBar=##
```

where ## is the number of characters wide that you would like the Source window. The default is SplitBar=27.

You must restart synario.exe for it to recognize the new value.

Text Editor and Report Viewer

You can select your preferences for the file modes, screen font, and customized key mappings from the Options menu in these programs. The online help discusses these options and the choices you have for changing them.

Schematic Editor, Symbol Editor, and Hierarchy Navigator

The Synario Schematic tools use special configuration files, one for each Device Kit that you have installed. You can change the settings in these files with the Synario INI Editor, including setting system colors and adding custom menus. All of these configuration files have an .ini extension and are installed in the Synario configuration directory (c:\synario\config is the default installation directory).

To run the INI Editor with the currently-selected device kit INI file loaded:

Select Options: Schematic in the Synario Project Navigator.

CAUTION *Changes you make to one INI file are not reflected in the other INI files. If you want the changes in all of the INI files, you must load the INI file for each device kit in turn and make the changes. For example, if you have a LCA4000 part selected and select Options:Schematic, the INI Editor opens with the file LCA4000.INI loaded. If you changed the Border Color for schematics in the LCA4000.INI file, that change would be seen only when you had a LCA4000 part selected. When you change to a different device kit, the INI file being used by the Schematic tools changes.*

See Also



Online help in the Synario INI Editor
Appendix C, "The Synario INI Editor"

Synario Project Navigator Configurable Menu

You can add a menu to the Synario Project Navigator to access other Windows programs. This user-configurable menu is controlled by an INI-format file named `synmenu.cfg` installed in the `synario\config` directory. If `synmenu.cfg` is not found, only the standard Project Navigator menus are displayed.

The name of the menu is determined by the section name (in square brackets) and each menu item is defined by the members of the section.

The basic format for defining the menu is:

```
[menu_name]
item_text=program,valid_extension,command_line,message_bar_text
item_text= . . .
```

Note *If you add an ampersand (&) in the text of the Menu Name or Item Text, the character following the ampersand is underlined, and the menu (or menu item) is available through the keyboard shortcuts (Alt+letter).*

The Message Bar Text should follow the format used by the fixed Synario menus.

Menu Name

menu_name

The Menu Name is the text that is displayed on the menu bar of the Synario Project Navigator between the Options and Windows menus. For example, if you enter

```
[&Tool]
```

a menu named "Tool" is inserted, accessed by Alt+T (underlined letter).

Menu Item

item_text

The Menu Item is the text for each menu item under the menu. Spaces are allowed.

Use & before a letter to allow keyboard access to the menu items (remember to give them unique letters).

Use a minus sign (-) at the beginning of the `item_text` line to place a separator line before that menu item.

For example, if you enter

```
-C&AD=program,valid_extension,command_line,message_bar_text
```

a menu item named CAD is inserted below a separator line, accessed with A (underlined letter).

Program *program*

The Program is the name of the program that is run when the menu item is selected. It may contain a complete path specification if the program is not in your PATH environment variable search string (in your autoexec.bat file).

To use the Synario path:

Use the escape sequence \$\$ to substitute the path of the Synario executable.

For example, if synario.exe is running in c:\synario, the program name \$\$\syndos.exe is translated to c:\synario\syndos.exe.

Extensions *valid_extension*

This field specifies which file extensions are supported by the program, and enables the menu when a source with a supported extension is selected

CAUTION *DO NOT USE a period to specify the extensions.*

For reference, the Synario source extensions are shown below:

Source	Extension
Project Notebook	SYN
Device	SYN
HDL Source	ABL
Schematic	SCH
Test Fixture	TF
All Files	*

You can selectively enable the menu item for documents (which appear under the Project Notebook) based on their extension. For example, if your project contains both Word documents (.DOC) and Xilinx memory files (.MEM), a menu item with a *valid_extension* of DOC is enabled only for Word documents and not Xilinx memory files.

CAUTION *The `valid_extension` field differs from the configurable menus in the Schematic Editor, which expect `*` in front of the extension and a blank field instead of a single `*` for the All Files options.*

Command Line Options

command_line

The Command Line field specifies filenames and switches that are passed to and run with the program name. The command line can use escape sequences to customize the switches based on the filename extension of the selected source. The substitution escape sequences are as follows, and the examples are based on a selected source with the full path of C:\SYNARIO\TEST\TAZ.ABL:

Sequence	Meaning	Example Result
\$\$	Escape for a single \$	\$
\$B	Base name	TAZ
\$D	Drive and directory	C:\SYNARIO\TEST
\$E	Extension only, including period (.)	.ABL
\$F	Base name plus extension	TAZ.ABL
\$N	Complete name	C:\SYNARIO\TEST\TAZ.ABL
\$R	Drive + directory + base name	C:\SYNARIO\TEST\TAZ
\$Z	Ignored (required for blank argument field)	

Note *The \$Z parameter is required if the program has no arguments.*

Note that the following escape sequences are equivalent:

\$F is equivalent to \$B\$E
 \$R is equivalent to \$D\\$B
 \$N is equivalent to \$D\\$B\$E

Message Bar Text

message_bar_text

The Message Bar Text field is an optional field for text to be displayed in the message bar when the user selects the menu item. If no text is specified, a default message of "User-defined menu item" appears on the message bar. The message text for the menu (for example, Tools) is always "User-defined menu" and cannot be changed.

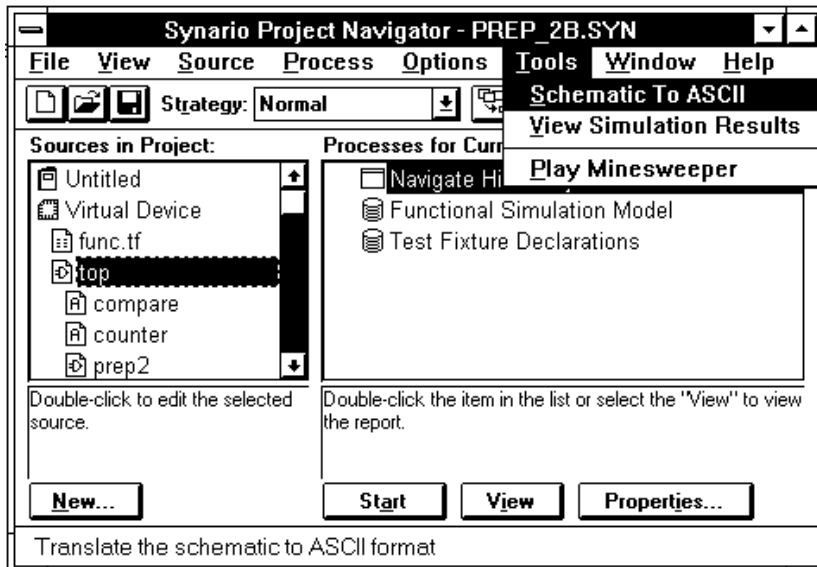
Note *The Synario Schematic Editor configurable menus do not support the Message Bar Text field.*

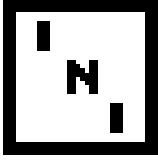
Examples Following is an example of a menu configuration file.

```
[&Tools]
&Schematic To ASCII=ascout.exe,sch,$F,Translate the schematic
to ASCII format
&View Simulation Results=e:\foo\dir\waves.exe,tf,-nav $F,View
the simulation waveforms
-&Play Minesweeper=winmine.exe,*,$Z,Hope the boss doesn't catch
you
```

These lines would produce a Tool menu similar to that shown in Figure B-1.

Figure B-1
User-defined Tool Menu





Appendix C

The Synario INI Editor

Synario's Schematic Editor, Symbol Editor, Hierarchy Navigator, and Waveform Viewer use an initialization file to control their appearance and operation, and to set the default values for many parameters (including pin, symbol, and net attributes).

There is a separate initialization file for each targeted device type. For example, if you select an LCA 3000 device as your target, Synario automatically uses the LCA3000.INI file. If you're using generic (retargetable) symbols and haven't yet selected a target device, Synario loads the RETARGET.INI file.

These initialization files are stored in the CONFIG subdirectory, directly beneath the main Synario directory (C:\SYNARIO, usually).

The INI Editor lets you view and alter the values of the parameters in the INI files. These parameters give you control over the following (and other) functions and parameters:

- ◆ Attributes
- ◆ Color assignments
- ◆ Display parameters (such as Show Border and Show Pin Dots)
- ◆ Default symbol type for the Symbol Editor
- ◆ Default text editor
- ◆ Global net names
- ◆ IC or PCB development mode
- ◆ Library search paths
- ◆ Printer configuration
- ◆ Schematic sheet size and layout parameters
- ◆ Tools and Processes menu entries

CAUTION *When editing an INI file, remember that there is a separate initialization file for each target-device family. For example, the LCA2000, LCA3000, and LCA4000 families each have their own INI file. If you work with more than one device family, you must edit the INI file for each family. Changes made in one INI file are not automatically applied to the other files.*

You can also use the INI Editor to customize the INI file for each family. For example, you might select different background colors to serve as a visual reminder of which family you're working with. Each INI file can also have a different set of Global attributes.

Saving Changes

After changing anything, use the Save command from the File menu to update the loaded INI file. If you decide to discard your changes, use the Exit command and select No ("Don't Save Changes") to quit without saving.

The BINARY.INI File

The Editors and Hierarchy Navigator often have to refer to the contents of the INI file. INI files are ASCII text files, and reading and interpreting an ASCII file takes much more time than directly loading the values.

Therefore, each time you save an INI file (or change your target device), the INI Editor automatically creates a *binary* version of the INI file (BINARY.INI) in the directory specified in your DOS TEMP environment variable. If you haven't defined a TEMP variable, the binary file is stored on your hard disk's root directory (usually C:\).

When Synario starts running, it looks for BINARY.INI in the TEMP directory (or, if there is no TEMP environment variable, in C:\). If it cannot find BINARY.INI, it automatically recreates the file.

When you alter an INI file and save the changes, the INI Editor asks if you want this file to be the INI file for the current project. If you are editing the current project's INI file, choose Yes. The INI Editor will update BINARY.INI. If it is not the current project's INI file, choose No.

Custom INI Files

You can use the Save As command from the File menu to save the INI file under any other name, to create customized INI files. However, since there is already a unique INI file for each target device, you would probably want to customize the existing INI files, rather than creating a new INI file with a different name.

Using the Synario INI Editor

In addition to the File menu, the INI Editor's menu bar displays the following:

- ◆ Controls
- ◆ Tools
- ◆ Attributes
- ◆ Search Paths

The settings and controls for each menu are explained in the following sections.

Controls Menu

System Controls

The system controls affect the general behavior of both Editors and the Hierarchy Navigator.

Application Mode

This parameter configures Synario for IC design or PCB design, or both. The default setting is IC.

IC Only	Provides instance-name support for ASICs.
PCB Only	Provides support for pin numbers and reference designators for board packaging.
BOTH IC & PCB	Provides support for instance names, pin numbers, and reference designators.

Bus Parentheses

The Bus Parentheses list box lets you select the delimiting character for indices on ordered bits. They can be enclosed in brackets, parentheses, curly braces, or angle brackets, as shown below. The default is brackets.

A[2], A[1], A[0]

A(2), A(1), A(0)

A{2}, A{1}, A{0}

A<2>, A<1>, A<0>

First Character Must Be Alphabetic

The first character of a net name or instance is usually a letter. (Numbers are usually suffixes to identify a specific instance or net.) If this check box is unmarked, the first character can be a number. This is the default setting.

Simulator

Configures Synario for a particular simulation environment. This parameter is set correctly for Synario, as shipped. Do not change it unless you have a specific reason for doing so (such as using a different simulator).

A major feature of the simulation environment is providing hardware description language (HDL) templates in the Symbol Editor and Hierarchy Navigator. This facilitates writing behavioral models for the simulators. It is also used in the Dynamic Waveform Interface.

Text Editor

Specifies the editor Synario uses to display text. The default is the Synario File Editor. If you want to use a different editor, enter its name here. If the alternate editor's directory does not appear in the DOS PATH statement, give the fully qualified pathname.

Display Controls

The Display Controls dialog box contains a group of 12 check boxes for the following display features. Checking a box displays or enables the corresponding feature. The first 8 items can be overridden (for the current session only) with the Display Options command from the Options menu in the Schematic or Symbol Editors.

Show Border

Turns on and off screen and plotter display of the schematic and symbol borders.

Show Pin Dots

Turns on and off the screen and plotter display of pin dots (unconnected pins). When off, it reduces repaint time and clutter in a crowded drawing.

Show Pin Numbers

Turns on and off the screen and plotter display of pin numbers. When off, it reduces repaint time and clutter in a crowded drawing.

Show Symbol Text

Turns on and off the screen and plotter display of fixed text inside symbols. When off, it reduces repaint time and clutter in a crowded drawing.

Show Symbol Attributes

Turns on and off the screen and plotter display of symbol attributes. When off, it reduces repaint time and clutter in a crowded drawing.

Show Solder Dots

Turns on and off the screen and plotter display of solder dots.

Show Off Page Connects

On multiple-sheet schematics, you can show references to other sheets at nets that connect across more than one sheet. The display is enabled on wire segments with their names at the end of the wire.

Show Open Ends

Wires not terminating on a net name flag, symbol pin, or another wire are considered schematic errors. This parameter controls whether they are highlighted on the screen and plotter.

Allow Rotated Pin Numbers

The numbers on the pins of a symbol are normally displayed as horizontal text. Optionally, the top and bottom pin numbers can be displayed as vertical text.

Allow Rotated Net Names

Controls whether net names at the ends of vertical wires are displayed vertically or horizontally.

Show Net Numbers

This parameter turns on and off the screen and plotter display of node numbers. It is useful for reducing the clutter on a crowded drawing. This parameter affects only the Hierarchy Navigator display.

Every node in the circuit has a node number assigned in the Synario database. These node numbers are used internally and can also be used by simulators like SPICE, which require numbers rather than names.

Show Simulation Values

Turns the screen and plotter display of simulation values on the schematic on and off. This feature is needed only when you view the results of a simulation.

Symbol Controls

These control two defaults in the Symbol Editor.

Default Symbol Type

Default Symbol Type is the type assigned to a newly created symbol. You can choose from Block, Cell, Component, Gate, Graphic, Pin, or No Default. These types are described in the list below. You can override the default type by selecting a different type using the Symbol Editor's Change Type command.

If this parameter is set to No Default, the Symbol Editor prompts you for the symbol type when you start a new drawing. Refer to the *Synario Schematic Editor Reference* for details about the different symbol types.

Block	Represents hierarchy
Cell	Primitive in IC design
Component	Primitive in PCB designs representing complete packaged device
Gate	Primitive in PCB designs representing a fraction of complete device (for example, one of four NAND gates in a 7400)
Graphic	Used for non-electrical information, such as tables and notes

Master	Used for title blocks and other non-electrical information positioned in a specific corner of the drawing
Pin	Represents physical pins on an edge connector or PCB

The Master type is not available as a default, only by prompting or from the Change Type command. It is assumed Master is the least-used symbol and users would create Master symbols only occasionally.

Default Pin Name Offset

Default Pin Name Offset controls the distance between a pin's name label and the pin itself. It is measured in quarter-grid units. The default is nine units. The value can range between 0 and 31.

Graphic Options

This dialog box sets the defaults for the Graphic Options dialog box in the Schematic and Symbol Editors. Any of these can be overridden in the Editor for the current working session.

Text Justification	Horizontal text can be left-justified, right-justified, or centered. This parameter applies to both fixed graphic text and text in attribute windows. Any change in this setting affects text that is added after the change, not existing text.
Text Size	There are three text sizes for drawing fixed graphic text and symbol attribute windows. The choices are Small, Medium, and Large, which correspond to 5, 7, and 9 fine (quarter) grids in height. The default is Small. Any change in this setting affects text that is added after the change, not existing text.
Vertical Text	Text is normally drawn horizontally. When the Vertical Text box is checked, text is placed vertically by default.
Grid Spacing	Grid Spacing controls the placement of graphic objects, <i>not</i> symbols or wires (which must always fall on the Primary grid). You can set the default increment to the Primary grid spacing, or to one-half or one-quarter that value. Smaller values allow more precise placement.
Show Grid	Enables the Primary grid display. The grid appears as an array of dots, with one dot at each grid intersection. Every tenth grid point is larger. As you "zoom out" and the grids get closer together, some grid dots may not be displayed.

Full Cursor	Selects the normal cursor (a small plus sign) or the full-screen cursor. Using the full-screen cursor makes it easier to align objects.
Wide Lines	When Wide Lines is checked, all graphic elements (<i>not</i> symbols or wires) are drawn with double-thick lines. These heavy lines have the same weight as schematic buses. Any change in this setting affects graphics that are drawn after the change, not existing graphics.

Sheet Layout

The Sheet Layout dialog box sets defaults for the border and the primary grid.

Zones

The border is divided into horizontal and vertical zones (sections) to simplify locating a specific item. For example, a flip-flop might be in the B7 zone, or an I/O marker in D3.

By default, the horizontal zones are numbered, the vertical lettered. When the “Draw Numbers on Vertical Axis” box is checked, the horizontal zones are lettered, the vertical numbered.

When the “Horizontal Zones Increase toward the Right” and “Vertical Zones Increase toward the Top” boxes are checked, the lowest numbers (or letters) are at the left and bottom. When these boxes are cleared, the lowest numbers (or letters) are at the top and right.

The Number of Zones edit boxes set the number of border divisions. The acceptable range of values is two to nine divisions.

Grid

The Grid Size and Grid Units settings are self-explanatory. Inches or centimeters at a 0.1 increment are the most-common choices. 0.1 inch is the default.

A symbol does not have an absolute size; it is scaled in grid units. Therefore, selecting a smaller grid may let you place more symbols on a schematic with a specific size. On the other hand, a larger grid will produce larger symbols when the schematic is printed.

Automatically Add Master Symbols

Master symbols are used for reference items that appear on every schematic, such as the project name, a title bar, or the company logo. If you want a Master symbol to be added to each schematic, type its file name in the Automatically Add Master Symbols edit box.

The symbol is automatically placed in the same corner of the schematic as its origin. For example, if the symbol's origin is at its lower-left corner, the symbol will be placed at the lower-left corner of the schematic.

To specify more than one Master symbol, separate the Master symbol filenames with spaces. (If they have the same origin, they will overlap.) As with other symbols, *do not* specify the path. The Editor will explore the Symbol Libraries search-path list for the first symbol file with a matching name. If this is not the Master symbol you want, select the desired symbol from the Add Symbol list box.

Sheet Sizes

The Sheet Sizes dialog box sets the permitted drawing sizes. The default sheet size is the size specified at the top of the list box. The sheet size for a particular drawing can be changed with the Sheet Setup command in the Schematic Editor. Typical sheet sizes are:

English (inches)	Metric (mm)
A = 11 8.5	A4 = 297 210
B = 17 11	A2 = 594 420
C = 22 17	A3 = 420 297
D = 34 22	A1 = 841 594
E = 44 34	A0 = 1189 841

The width is the first number. Since schematics are usually wider than they are high, the width is usually greater than the height. Height and width are measured in the Grid Units specified in the Sheet Layout dialog box (inches, centimeters, or millimeters). The maximum supported dimension is 8000 Grid Units.

To add a new size, click on the Add button. A new entry with the designation New and a length and width of zero is added to the list. Press TAB to select the edit boxes (or click on them), then change the Sheet Size, Width, and Height to the values you want.

To change the size of an existing sheet, click on the list-box line with its description. The values are copied to the edit boxes, where you can alter them.

To delete a sheet, click on its description. Then click on Delete.

The Move Up and Move Down buttons rearrange the listing. The Move Up button swaps the highlighted sheet with the sheet above it. The Move Down button swaps the highlighted sheet with the sheet below it.

Note *The sheet-size names are arbitrary and have no relation to either European paper sizes or American drafting paper sizes. You can use any letter, number, or name you want.*

Wave Controls

This dialog box controls the overall appearance of the Waveform Viewer. Colors are set in the Wave Colors dialog box.

Padding Around Text	The padding above and below text in the waveform name area. This adjusts the space available for the waveform display. Padding is in units of one-quarter the text height.
Gap Between Waveforms	The space between waveforms. It's in units of one-quarter the text height.
Characters in Name Field	The number of characters visible in the waveform name area.
Reverse Bits in Bus	If Yes, the least-significant bit (LSB) of buses is displayed to the left whenever the bus value is displayed as a number. If No, the bus value is displayed with the same bit order as a binary number (the LSB to the right).
Bus Radix	The radix, or base, used to display buses. The choices are Binary, Octal, Decimal, and Hexadecimal.

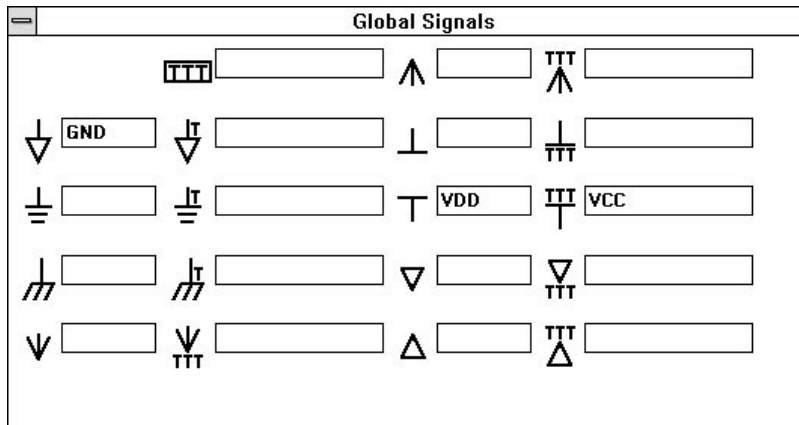
Global Nets

Global nets are net names that have predefined symbols associated with them. When one of these global names is assigned to a net (GND, for example), the corresponding symbol is attached to the net in your schematic.

Global signals can be accessed across all hierarchy levels and across all sheets and schematics in a design. For this reason, names assigned as global net names cannot be used as "local" net names.

The available symbols are shown in Figure C-1. A symbol cannot be used until a name has been assigned to it. (Three of the symbols are already named and can be used immediately.) Click on the edit box next to the symbol and type in the name you want. You can change or remove a name the same way.

Figure C-1
Global Signals Dialog Box



There are three types of global net symbols.

Labeled Symbols The symbols in columns 2 and 4 of Figure C-1 are *labeled* symbols. The name you assign to one of these symbols is attached to the symbol to label it, replacing the "T" or "TTT."

You can assign more than one name to these symbols. Multiple names are separated with a space (not a comma).

Unlabeled Symbols The symbols in columns 1 and 3 of Figure C-1 are *unlabeled* symbols. The name you assign is *not* shown on the symbol. The symbol is the only visible indication of the net name. Use the Query command to view the name.

You can assign only one name to an unlabeled symbol. This limitation is to prevent confusion, since there is no visible indication of the assigned name.

No Symbol You can assign a name to the box with "TTT" at the top of the second column. This name is global. The name is attached to the net, but other than the box surrounding it, there is no symbol.

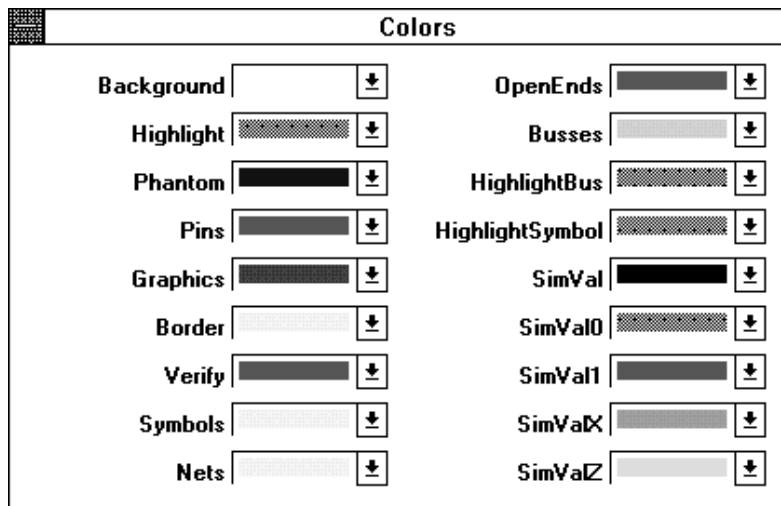
Unnamed symbols can have only one name associated with each symbol. Named symbols and the No Symbol icon can have multiple names. Multiple names are separated by spaces.

Note Global ground symbols are drawn only at the bottom of vertical wires, while global supply symbols are drawn only at the top of vertical wires. If the wire is not vertical, the global symbol is not drawn and its name is displayed inside a box.

Colors

The Colors dialog box assigns colors to various Symbol and Schematic Editor display functions. There are 18 combo boxes, one for each Editor display function that can have a unique color.

Figure C-2
Colors Dialog Box



To change a color, click on the arrow in the combo box, then click on the desired color in the color-bar display that appears. The display functions are listed below.

- | | |
|------------|---|
| Background | The background color of the Editor window |
| Highlight | Highlighted nets |
| Phantom | Highlight group-editing functions |
| Pins | Symbol pin dots (unconnected pins) |
| Graphics | Lines, boxes, circles, arcs, text |

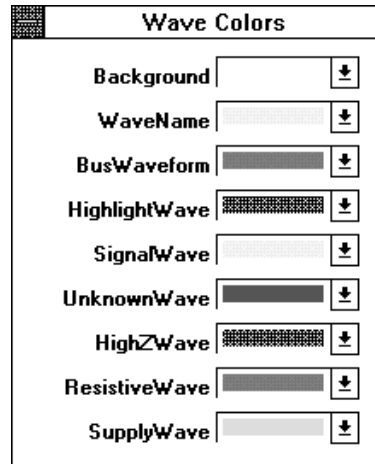
Border	Sheet border
Verify	Indicates connection to net
Symbols	Symbol body
Nets	Wires and net names
OpenEnds	Dot on hanging wire or isolated net name
Buses	Buses and bus names
HighlightBus	Highlighted buses
HighlightSymbol	Highlighted symbols
SimVal	Text showing simulation value on schematic
SimVal0	Small square on schematic indicating logic low at probed node
SimVal1	Small square on schematic indicating logic high at probed node
SimValX	Small square on schematic indicating unknown state at probed node
SimValZ	Small square on schematic indicating high impedance at probed node

Note *Don't assign the Background color to any other display function. That function will then be invisible against a background with the same color.*

Wave Colors

The Wave Colors dialog box assigns colors to various Waveform Viewer display functions. There are nine combo boxes, one for each Waveform Viewer display function that can have a unique color.

Figure C-3
Wave Colors Dialog Box



To change a color, click on the arrow in the combo box, then click on the desired color in the color-bar display that appears. The display functions are listed below.

Background	The background color of the Waveform Viewer window
WaveName	Display of waveform names
BusWaveform	Display of bus waveforms
HighlightWave	Color of selected (highlighted) waveform
SignalWave	Display of single-bit signals
UnknownWave	Buses or signals displayed with unknown value
HighZWave	Display of high-impedance signals
ResistiveWave	Display of resistive signals
SupplyWave	Display of low-impedance (direct from Vdd) signals

Note Don't assign the Background color to any other display function. That function will then be invisible against a background with the same color.

Tools Menu

You can run other programs from within the Schematic Editor, the Symbol Editor, and the Hierarchy Navigator. These new programs are listed in the Tools menu of the Editors and the Navigator, and in the Navigator's Processes menu. (If no programs are assigned, the Tools and Process menus are not displayed.)

There are three dialog boxes for adding programs to the Tools menus, and a fourth dialog box for adding programs to the Process menu in the Hierarchy Navigator. The only distinction between Tools and Processes in the Hierarchy Navigator is that the Processes menu is intended for programs that create netlists. You can ignore this distinction if you wish.

All four dialog boxes are identical and work the same way. The Hierarchy Navigator Process Menu dialog box is shown in Figure C-4.

Figure C-4
Hierarchy Navigator Process Menu Dialog Box



- ◆ The Menu Label is the text string that appears in the menu to identify the added tool or process. You click on this menu item to run the new program.
- ◆ The Application is the filename of the program. (You can also specify DOS batch (.BAT) files.) If the program is not in the working directory or one of the directories in the DOS PATH statement, enter the fully qualified path name.
- ◆ The Flags are any command-line switches or options needed.

You can fill in these edit boxes by hand, or you can use the Add button to browse the programs on your hard disk. The Add button displays the generic Open File dialog box, labeled Choose Application. Find the application you wish to add, then double-click on it. (Or highlight it and click on OK.) The Menu Label and Application edit boxes are automatically filled in.

When you use the Add button, the path is not included in the Application edit box. If the correct path is not part of the DOS PATH statement, you must add the full path to the application's name. You must also fill in any needed command-line flags or switches.

The list box below the edit box displays the tools or processes in the same order they will appear in the menu. The Move Up and Move Down buttons rearrange the listing. The Move Up button swaps the highlighted application with the tool above it. The Move Down button swaps the highlighted application with the application below it. Delete an application by highlighting it then clicking the Delete button.

Symbol Tools

Any utilities, netlisters, or other processes that do not require the Navigator to operate (such as the Notepad and archiving tools) can be added to the Tools menu of the Symbol Editor. The Verilog and VHDL netlisters are the only tools shipped with Synario that currently work in the Symbol Tools menu.

Schematic Tools

Any utilities, netlisters, or other processes that don't require the Navigator to operate can be added to the Tools menu of the Schematic Editor. The Verilog and VHDL netlisters are the only tools shipped with Synario that currently work in the Schematic Tools menu.

Navigator Tools

This dialog box adds entries to the Tools menu of the Hierarchy Navigator. These entries spawn tasks for the Waveform Viewer and the Design Analysis Tools (DAT). You can write your own interface programs and add them to the Tools menu.

Processes

This dialog box adds entries to the Processes menu of the Hierarchy Navigator. These entries spawn tasks for netlisting or simulation. Synario supports several netlisters that are accessed through this menu. In addition, you can write your own interface programs and call them from the Processes menu.

***Note** The distinction between the types of programs added to the Navigator's Tools and Processes is arbitrary, and designed only to make it easier to find a specific tool. You can place your own entries in either menu.*

Attributes Menu

Attributes are created with the INI Editor.

Symbol, Pin, and Net Attributes

There are separate dialog boxes for editing symbol, pin, and net attributes. All have a list box showing the current attribute definitions, and an edit box at the top where an attribute name can be added, deleted, or altered.

The Symbol and Pin dialog boxes include four radio buttons to select the attribute modifier. The Symbol dialog box also has a second edit box for entering an attribute window number. The Symbol dialog box is therefore shown in Figure C-5, because it includes all the features of the Attributes dialog boxes.

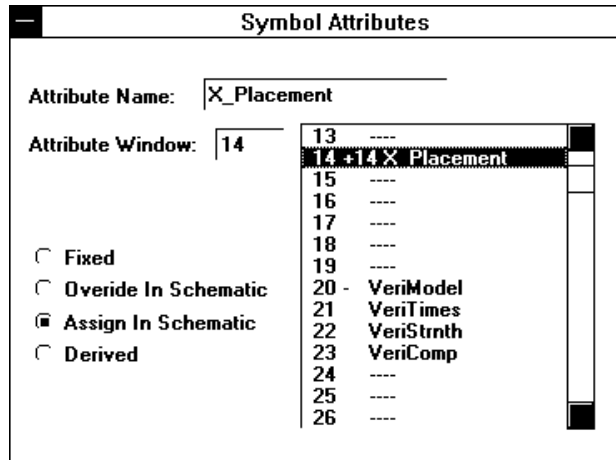
Attribute Data Fields

Each attribute has three or four data fields, but you don't always have to enter a value in every field.

First Field

The first field is the attribute number. The attribute is represented in the attribute database by this number, and Synario uses this number to access it.

Figure C-5
Symbol Attributes Dialog Box



Unused attribute numbers are shown with four dashes in the name field. New attributes can be added to these empty fields. Attributes 00 through 99 are pre-defined and should not be changed (even those with blank name fields). You can define attributes 100 through 199 in any way you like.

Second Field

The second field is the attribute modifier. The modifier controls where and how attribute values can be entered or altered. The default is a blank (no entry).

The modifiers are described below.

Attribute Modifiers	Description
<blank>	<p>“Override in Schematic” If an attribute has been assigned a value in the Symbol Editor, the value can be changed in the Schematic Editor and the Hierarchy Navigator. If it was not assigned a value in the Symbol Editor, it <i>cannot</i> be assigned a value in the Schematic Editor or the Hierarchy Navigator.</p> <p>This field can be modified for pin and symbol attributes, but not net attributes.</p>
-	<p>“Fixed” The attribute value can be modified only in the Symbol Editor; it cannot be overridden in the Schematic Editor or Hierarchy Navigator.</p>
+	<p>“Assign in Schematic” The attribute value can be assigned or modified in the Symbol Editor, Schematic Editor, or the Hierarchy Navigator. If you change the value in more than one place, values changed in the Schematic Editor override those changed in the Symbol Editor and values changed in the Hierarchy Navigator override all others.</p>
*	<p>“Derived” Specifies a derived attribute you can modify in the Symbol Editor, and override in the Schematic Editor or the Hierarchy Navigator.</p>

There is a fifth modifier, the exclamation point (!), which does not appear as one of the modifier radio buttons in the Attribute Editor dialog boxes.

!	<p>"System" The attribute value can be edited only with special Synario commands such as Instance Name. Most system attributes are for the internal use of the Editors and Hierarchy Navigator. (A few of them—such as Instance Name—can be viewed and modified by the user.) You cannot assign the System modifier to an attribute.</p>
---	--

Third Field

The third field is optional and specifies the attribute window. The attribute's value is displayed on or near the symbol in this window. Attribute windows are used only with symbol attributes. (Attribute windows are described in the *Synario Schematic Editor Reference*).

Fourth Field

The last field is the attribute name. You can enter your own names for the user-defined attributes (100–199). You can change the names of the system attributes (00–99) if you wish, because the association between an attribute and its value is made with the attribute's number, not its name.

Modifying Attributes

Use the following procedure to add or modify a symbol attribute. The procedure is nearly the same for pin and net attributes; skip the steps that don't apply to the selected attribute.

1. Click on the attribute you want to modify. You can select any line in the list box, even if the attribute number is the only field filled.
2. The attribute's name appears in the Attribute Name edit box. Type the new or changed attribute name.
3. Click on the appropriate radio button to assign the desired attribute modifier. "Override In Schematic" is the default modifier. If this modifier is selected, the modifier field is left blank in the list box. (This feature does not apply to net attributes.)
4. If you want the attribute displayed, enter a number in the Attribute Window edit box. Window numbers range from 00 to 99, and have no relation to the attribute numbers 00 through 99. (The attribute window feature does not apply to pin or net attributes.)

If two (or more) attributes use the same attribute window, the lowest-numbered attribute *that has a value* is displayed.

Example Attributes

Tables C-1, C-2, and C-3 show example attribute definitions for net, pin, and symbol attributes. Attribute modifiers, numbers, and windows are shown where appropriate.

The *Synario Schematic Editor Reference* has a more detailed discussion about using attributes and creating new ones.

Table C-1
Standard Net Attributes

Att Number	Att Mod	Attribute Name	Description
0	!	NetName	Net name
3		Cap	Capacitance
5		Length	
8		Width	
10		VeriType	Verilog Net Type
30		VHDLNetType	VHDL Net Type
31		VHDLBusType	VHDL Bus Type

Table C-2
Standard Pin Attributes

Att Number	Att Mod	Attribute Name	Description
0	–	PinName	pin name
1	–	Polarity	In, Out, BiDir
2		Fanin	Dimensionless number for IC loads
3		FanOut	Dimensionless number for IC drive
4	–	PinNumber	Used in PCBs for Gate or Component pin numbers; represents physical pin connection
5		WireOr	Tristate, Opencollector, or Yes
6	–	PinGroup	Used in PCB design; indicates can swap pins

Att Number	Att Mod	Attribute Name	Description
7		LoadLow	Current load in low state (μA)
8		DriveLow	Current drive in low state (μA)
9		OpenOK	OK to be unconnected pin
10	-	SilosName	Identifies pins in models
11		SilosLoad	Numeric load factor for load calculation
14	-	VeriName	Alternate pin name or order for pins
15		LoadHigh	Current load in high state (μA)
16		DriveHigh	Current drive in high state (μA)
18	-	TimilName	Alternate pin name or order for pins
20		LoadCap	Capacitive load (pF)
21		DriveCap	Capacitive drive (pF)
22	+	KCL	Drive factor for simulation models
23	+	Pin2Pin	Pin-to-pin delay for simulation models
24	+	DelayBack	Back-annotated delay for simulation models
25	+	ChkPulseW	Check for pulse width violation on this pin
26	+	ChkHold	Check for hold time violation on this pin
27	+	ChkSetup	Check for setup time violation on this pin
30		VHDLPinType	Port type if scalar port in VHDL
31		VHDLBusPinType	Port type if vector port in VHDL
32		VHDLDefValue	Default value for Port
33		VHDLNetConv	Type conversion function for port
34		VHDLBusConv	Type conversion function for port
35		VHDLPinUse	Used for port type of BUFFER in VHDL
36		SpiceOrder	Integer that forces order of subcircuit pins
40		HiLoPinName	Alternate pin name used for HiLo simulations

Att Number	Att Mod	Attribute Name	Description
50		XSimPinName	Pin name used for X-Sim primitives
51		PinNegation	Apply negation to this pin
90	-	BusPin_A	First set of pins attached to bus pin
91	-	BusPin_B	Second set of pins attached to bus pin
92	-	BusPin_C	Third set of pins attached to bus pin
93	-	BusPin_D	Fourth set of pins attached to bus pin
94	-	BusPin_E	Fifth set of pins attached to bus pin
95	-	BusPin_F	Sixth set of pins attached to bus pin
96	-	BusPin_G	Seventh set of pins attached to bus pin
97	-	BusPin_H	Eighth set of pins attached to bus pin

Table C-3
Standard Symbol Attributes

Att Number	Att Mod	Attribute Window	Attribute Name	Description
0	!	0	InstName	Instance Name
1	!	1	Type	Symbol Name
2	!	2	RefDes	Reference designator for PCBs
3		3	Value	General value parameter
4			PartNum	PCB part number
5			PartShape	Footprint of PCB part
6			CompName	Specifies DeMorgan-equivalent gates with <i>same</i> name
8			Prefix	SPICE element prefix (Q, M, R ...)
9			TreeStop	Used to split large designs
10	-		SilosModel	Primitive model name for SILOS
11			SilosTimes	Delay specifier for SILOS primitives
15	-		TegasModel	Tegas model parameter
17	-		XSimModel	X-Sim primitive model name
18	+		BodyDelay	Body delay parameter for simulation
20	-		VeriModel	Primitive or model name for Verilog
21			VeriTimes	Delay specification for Verilog
22			VeriStrnth	Strength specification for Verilog
25	-		TimilModel	Primitive or Model name for Timemill
26			TimilExtra	Extra parameter for Timemill
34			Impedance	SPICE parameter
35			Width	SPICE transistor width

Att Number	Att Mod	Attribute Window	Attribute Name	Description
36			Length	SPICE transistor length
37			Multi	Multiplication factor for SPICE
38			SpiceModel	Model card for SPICE
39			SpiceLine	Model parameters for SPICE
40			SpiceLine2	Additional SPICE model parameters
41	*		AreaS	Area of source for SPICE
42	*		AreaD	Area of drain for SPICE
43	*		PeriS	Perimeter of source for SPICE
44	*		PeriD	Perimeter of drain for SPICE
45			NRS	Squares of source diffusion, SPICE
46			NRD	Squares of drain diffusion, SPICE
47			DefSub	Substrate node
60	-		GND	*Power connection attribute for PCBs; typ GND
61	-		VDD	*Power connection attribute for PCBs; typ VDD
62	-		VCC	*Power connection attribute for PCBs; typ VCC
63	-		PCBGlobal3	*Power connection attribute for PCBs
64	-		PCBGlobal4	*Power connection attribute for PCBs
65	-		PCBGlobal5	*Power connection attribute for PCBs
66	-		PCBGlobal6	*Power connection attribute for PCBs
67	-		PCBGlobal7	*Power connection attribute for PCBs
68	-		PCBGlobal8	*Power connection attribute for PCBs
69	-		PCBGlobal9	*Power connection attribute for PCBs

Att Number	Att Mod	Attribute Window	Attribute Name	Description
70	-		HiloModel	Specifies model for HILO primitives
71			HiloTimes	Specifies HILO delay times
72			HiloStrength	Specifies HILO driving strength
73			HiloParam	
74			HiloParamValue	
75			HiloDelayScale	
76			HiloVisibility	
78			VHDLConfig	
79			VHDLUseLib	
80			VHDLModel	
81			VHDL1	User-definable for VHDL
82			VHDL2	User-definable for VHDL
83			VHDL3	User-definable for VHDL
84			VHDL4	User-definable for VHDL
85			VHDL5	User-definable for VHDL
86			VHDL6	User-definable for VHDL
87			VHDL7	User-definable for VHDL
88			VHDL8	User-definable for VHDL
89			VHDL9	User-definable for VHDL
99			EllaType	

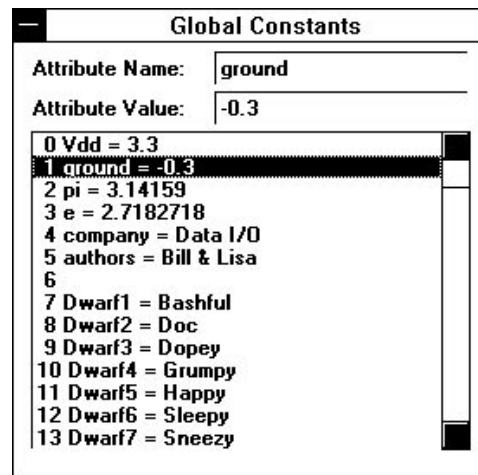
Global Attributes

The Global Constants dialog box defines the global attributes in schematics. There are 20 global attributes, all of which are user defined. They can be used to define anything, but are most commonly used for attributes that apply to all schematics, such as supply voltage (Vdd) or design-rule dimensions.

1. Click on the desired global attribute number from the list box.
2. Type the name of the global attribute in the Attribute Name edit field.
3. Type the value of the global attribute in the Attribute Value edit field.

Global attributes can be modified in the Hierarchy Navigator with the Edit Constants command. Any changes are discarded when you exit the Navigator.

Figure C-6
Global Attribute Editor



Search Paths Menu

Project, Model, and Symbol Libraries

These dialog boxes let you modify the search paths for Projects, Models, and Symbols. The directories are searched in the order they appear in the list.

When you highlight a name, it appears in the Path edit box where it can be modified. The INI files supplied with Synario prefix all paths with the %Root environment variable. (This variable is the path of the main Synario directory—C:\SYNARIO, usually.)

You do not have to use the %Root variable in any paths you add. You can enter a fully-qualified path. Or, since Synario has access to all DOS environment variables, you can add your own environment variables to AUTOEXEC.BAT and include them in the path.

Enabling and Disabling Paths

A plus sign (+) next to a path means it is enabled and will be searched. A minus sign (-) means the path is disabled and will not be searched. To enable or disable a path, click on its name to highlight it. Then click the Enable check box to mark or clear it.

Adding and Deleting Paths

To add a search path, click the Add Path button. The generic Open File dialog box appears (though it will be labeled Select Project Library, Select Symbol Library, or Select Model Library). Use the controls to select the directory you want. Click on the name of a file in that library, then click OK. (Or just double-click the name.) The directory containing this file is added at the bottom of the list.

The Delete button removes the path that is currently highlighted. The Move Up button swaps the highlighted path with the path below it; Move Down swaps the highlighted path with the path above it.

CAUTION *These search variables are properly configured when Synario is installed. Don't modify the paths or their order unless you have a specific reason for doing so (such as adding new directories with your own symbols).*

Libraries and Directory Structures

Libraries are collections of symbols, models, or hierarchical design blocks that can be accessed by the schematics of any design. Libraries are stored in directories other than project directories. A common directory for circuit elements simplifies design organization and makes it easier to ensure that all symbols and models are updated properly.

Program Directories

All software and related files used by Synario are located in a master directory called SYNARIO. (You can, however, select a different directory during installation.)

The files and directories in the master Synario directory are:

SYNARIO	Main directory; contains the Synario executable and help files. You can choose a different name during installation.
\CONFIG	Holds the device-specific INI files, configuration (.CFG) files, and licensing files.
\EXAMPLES	Contains Synario sample designs.
\LCA4000, \LIB5, \MAXPLUS, etc.	Contain device-specific symbols. Specific libraries depend on which fitters are installed.
\RETARGET	Contains retargeting symbols.

User Directories

Synario expects the source files for a particular design to be kept in a single directory (that is, the project directory). Whenever you start a new project, Synario prompts you for the name of that directory. The most common way to organize designs is to create a separate directory for each one.

Library Directories

Libraries store building blocks that can be reused in different designs. A library usually contains related items. For example, a symbol library might consist of symbols for 7400-series TTL devices. Another library might contain symbols for gate-array primitives.

Libraries are typically maintained by a system administrator, and the individuals who use the libraries are not allowed to modify them. This enforces consistency and prevents unauthorized changes.

Synario interfaces with three different types of libraries:

- ◆ Project directories
- ◆ Symbol libraries
- ◆ Model libraries

You can specify different libraries of each type and can control which ones are on the library search path. Please refer to the "Search Paths" section earlier in this chapter.

Project Directories

In Synario, a project directory contains *all* the files needed for a complete hierarchical design (except for primitive symbols). When an existing (non-symbol) file is imported from another directory, a copy of that file is placed in the project directory to guarantee its availability.

Model Libraries

Model libraries contain schematics that represent higher-level primitives, or behavioral-description (HDL) files of such primitives, or both. If the model library is included in the library search path, the Hierarchy Navigator will be able to display a lower-level or more detailed view of the circuit.

For example, a schematic containing logic gates is drawn and simulated using a gate-level simulator. A model library exists containing the transistor-level schematics of all the logic gates. A netlist for a switch-level simulation is easily obtained by adding the model library to the library search path and then running the netlister.

Symbol Libraries

Symbol libraries contain the primitive symbols for IC and PCB designs. Typical primitive symbols are PMOS and NMOS transistors, AND gates, NOR gates, or a 74ALS193 counter chip. Symbols are used throughout the hierarchy and at the primitive level in a design.

Library Searching

Synario searches for symbols in a fixed order. Synario uses the first symbol it finds with the required name. For example, a symbol *myname* can exist in a project directory and another version of *myname* can exist in a symbol library. Because project directories are searched before symbol libraries, the version of *myname* from the project directory is used.

The symbol search order is as follows:

1. The current project directory.
2. The project directories specified in the INI Editor's Project Libraries dialog box. Directories closest to the top of the list box are searched first.
3. The symbol libraries specified in the INI Editor's Symbol Libraries dialog box. Libraries closest to the top of the list box are searched first.

WARNING *If you open a schematic and the symbol search paths are not set correctly, the Schematic Editor may not be able to locate the correct symbols. If this happens, you'll see the wrong symbols, or blanks where the missing symbols should be.*

A schematic with missing symbols is opened with the name UNTITLED, rather than its original name. If you accidentally save the file, the original version won't be destroyed.

Synario also searches for schematics in a fixed order. It uses the first schematic it finds with the required name. The schematic search order is as follows:

1. The current project directory.
2. The project directories specified in the Synario INI Editor. Project directories closest to the top of the list box in the Synario INI Editor are searched first.
3. The model libraries specified in the Synario INI Editor. Model libraries closest to the top of the list box in the Synario INI Editor are searched first.

The following is a typical symbol search path.

```
%Root\sym_libs\std
%Root\sym_libs\misc
%Root\sym_libs\ttl
%Root\sym_libs\ttl_ls
%Root\sym_libs\ttl_als
%Root\sym_libs\ttl_as
```


Miscellaneous Control Options

The following parameters appear in the [Options] section of SYNARIO.INI, but do not appear in any of the INI Editor's dialog boxes. You can change them manually with a text editor. The INI Editor will not modify or overwrite your changes.

- | | |
|---------------|---|
| AttributeCaps | If set =Yes, attribute values are forced to uppercase. If set =No, attribute values retain the case in which you entered them. The default setting is No. |
| NetNameCaps | If set =Yes, net names are forced to uppercase. If set =No, net names retain the case in which you entered them. The default setting is No. |



Appendix D

PLD JEDEC Simulation

This chapter describes how to simulate the JEDEC file for PLD designs, including the MACH2 and Atmel devices. For other devices, and for functional simulation of any design, see the *Synario Simulator User Manual* if you have this option. This chapter includes the following topics:

Note *Currently, the MACH4 devices do not support JEDEC simulation; however, you can simulate the PLA file with the PLA simulator, which operates the same as described for JEDEC simulation.*

- ◆ How Synario simulates JEDEC files
- ◆ Report and trace types and break points
- ◆ Simulation and designs with buffered outputs
- ◆ Simulation and unspecified inputs
- ◆ Simulation and designs with feedback
- ◆ Register preloads in the simulator
- ◆ Test vectors and simulation
- ◆ Debugging state machines
- ◆ Multiple test vector sections
- ◆ Using macros and directives to create test vectors
- ◆ Don't cares in simulation
- ◆ Preset, reset and preload registers
- ◆ Asynchronous circuits

How Synario Simulates JEDEC Files

The Simulator Model

Using the JEDEC and device files, the JEDEC simulator builds a model of the design that includes macrocells, sum-terms, and product terms. In the Simulate JEDEC File properties, select the Report Type Macro-cell to display the model.

JEDEC and .tmv Vectors

JEDEC vectors include only test conditions for pins; the .tmv file vectors allow testing of internal nodes. Also, the vectors in the .tmv file can have different values for input than for output. For example, the .tmv file allows you to apply a 0 to a bidirectional pin that is an input before the clock and test for an H after the clock. A JEDEC vector could only have the H. Enable the Use .tmv File Test Vectors property to use the .tmv file for simulation in place of the JEDEC vectors.

JEDEC Vector Conversion

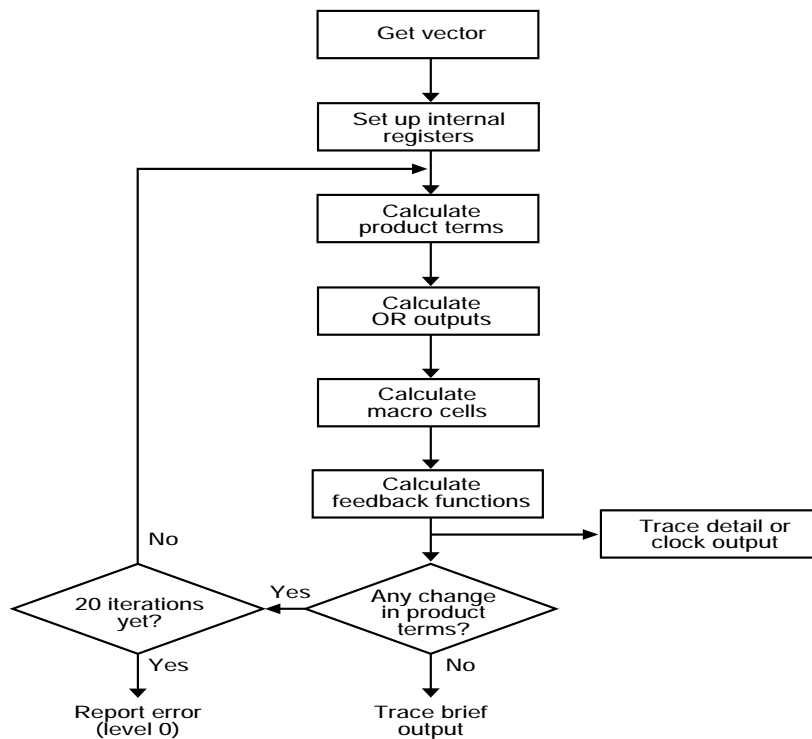
Internally, the JEDEC simulator uses the same test_vector format as the PLA file vectors, so it converts the JEDEC vectors to PLA format. When reading JEDEC test vectors, the simulator copies the H, L and Z into the output vector, and all test conditions into the input vector. It also makes the following conversions:

H	Converted to 1.
L	Converted to 0.
.Z.	Converted to 1 or the user-specified value.
.X.	Converted to 0 or the user-specified value.
.C.	Expanded to three vectors with .C. taking on the values 0, 1, and then 0. Use a Trace Type of Clock to observe the clock conversions.
.U.	Expanded to two vectors taking on the values 0 and then 1. Use a Trace Type of Clock to observe the clock conversions.
.K.	Expanded to three vectors with .K. taking on the values 1, 0, and then 1. Use a Trace Type of Clock to observe the clock conversions.
.D.	Expanded to two vectors taking on the values 1 and then 0. Use a Trace Type of Clock to observe the clock conversions.

JEDEC Simulation Flow

Figure D-1 shows a flow diagram of JEDEC simulation during evaluation of the inputs to the output. The JEDEC simulator applies the first test vector and performs any setup of internal registers (within the PLD) that results from the vector applied to the inputs. The simulator then calculates the product terms that result from the test vector, the OR outputs that result from the product terms, any macrocell outputs that result from the OR outputs, and then any feedback functions. The results of the simulator calculations are written to the .sim file.

Figure D-1
Simulation Processing Flow Diagram



The outputs of devices with feedback may require several successive evaluations until the outputs stabilize. After the feedback paths have been calculated, the JEDEC simulator checks to see if any changes have occurred with the device since the product terms were last calculated. If changes have occurred due to feedback functions, the calculations are again repeated. This iterative process continues until no changes are detected, or until 20 iterations have taken place. If 20 iterations take place and there are still changes, the design is determined to be unstable and an error is reported. More detailed information on simulating devices with feedback, and other advanced uses of the simulation program are presented in later in this appendix.

Report and Trace Types and Break Points

Report and trace types and break points allow you to control the amount of information that JEDEC simulation provides. JEDEC simulation can provide simple error messages (indicating that the actual outputs differ from the outputs you predicted in your test vectors), or detailed information about the states of internal registers and product terms of a device during simulation. If you simulate a design with Report Type set to None, you can determine whether any errors exist. If there are errors, then you can use the more detailed Report and Trace types to increase the amount of information provided until you have enough information to solve the problem. Examples of the output produced by the different Report and Trace types are given below.

With small designs, you can rerun JEDEC simulation with different Report and Trace types to obtain the information you need. With a larger or more complex design with many test vectors, however, reports may contain so much information that you have difficulty finding information on the error. With break points, you can target the error and limit the amount of information produced.

For example, you run a simulation and detect an error in the twentieth test vector out of 50 vectors. You want to see more information to determine the cause of the error, so you can rerun JEDEC simulation with a more detailed Trace Type for only vector 20 using breakpoints.

1. Simulate designs at Report Type: None to determine the existence of errors.
2. Once an error is found, increase the Trace and Report Types until you have enough information to correct the error.
3. Use break points to limit simulation results.

Each of the Report and Trace Type properties are discussed below.

Trace Type: Brief

Figure D-2 shows a brief trace simulation with a Tabular report output for the same source file that produced Figure D-6. Brief trace is the default.

Figure D-2

Brief Trace Simulation Output

```

Simulate SYNARIO 1.00  Date Tue Aug 14 10:56:35 1993
Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
      I
      C  N
      l O I D D D   F F
      k E T 1 2 3   1 2

V0001  C 0 0 1 1 1   H L
V0002  C 0 0 0 0 1   H L
V0003  C 0 1 1 1 1   L H
V0004  0 0 0 0 0 1   L L
4 out of 4 vectors passed.

```

Trace Type: Clock

Trace Type: Clock provides information similar to that of Brief, except that the Clock output displays the device inputs and outputs for each clock pulse. Figure D-3 shows that the output is expanded over that shown in Figure D-2 to show each clock pulse of the simulate operation.

Figure D-3

Clock Trace Simulation Output

```

Simulate SYNARIO 1.00  Date Tue Aug 14 10:58:41 1993
Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
      I
      C  N
      l O I D D D   F F
      k E T 1 2 3   1 2

V0001  0 0 0 1 1 1   H L
      1 0 0 1 1 1   H L
      0 0 0 1 1 1   H L
      ...edited...
V0004  0 0 0 0 0 1   L L
4 out of 4 vectors passed.

```

Trace Type: Detailed

Trace Type: Detailed provides information similar to that described for Clock, except that the Detailed output shows the device inputs and outputs for each iteration of the simulator. Figure D-4 shows the Detailed output for the same source file used to generate Figure D-2.

The output is expanded over that shown in Figure D-3 to show each iteration of the simulate operation that takes place to stabilize the device output. In this case, Vector 3 takes an extra iteration of the high clock pulse to stabilize.

Figure D-4
Detailed Table Format Simulation Output

```

Simulate SYNARIO 1.00  Date Tue Aug 14 10:59:21 1993

Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
      Data I/O Corp.      31 July 1993

      I
      C  N
      l  O  I  D  D  D   F  F
      k  E  T  1  2  3   1  2

V0001  0  0  0  1  1  1   Z  Z
        0  0  0  1  1  1   H  L
        1  0  0  1  1  1   H  L
        0  0  0  1  1  1   H  L
V0002  0  0  0  0  0  1   H  L
        1  0  0  0  0  1   H  L
        0  0  0  0  0  1   H  L
V0003  0  0  1  1  1  1   H  L
        1  0  1  1  1  1   L  L
        1  0  1  1  1  1   L  H
        0  0  1  1  1  1   L  H
V0004  0  0  0  0  0  1   L  L
4 out of 4 vectors passed.
    
```

Report Type: None

Figure D-6 shows the output of JEDEC simulation created during processing of regfb.abl (Figure D-5) with the Report Type set to None.

Figure D-5**Regfb.abl Source File**

```

module regfb
  title 'Operation of the simulator on devices with feedback
        Data I/O Corp.      31 July 1993'

        FB2      device 'P16R4';

        Clk,OE      pin 1,11;
        INIT,D1,D2,D3  pin 2,3,4,5,;
        F1,F2      pin 14,13;

        F1 istype 'reg_D,invert';

equations
  F1.D      = D1 & INIT;
  F2      = D2 & F1.Q;

  F2.OE     = D3;
  F1.C      = Clk;
  F1.OE     = !OE;

test_vectors  ([Clk,OE,INIT,D1,D2,D3] -> [ F1, F2])
  [.C., 0, 0 , 1, 1, 1] -> [ 1 , 0 ];
  [.C., 0, 0 , 0, 0, 1] -> [ 1 , 0 ];
  [.C., 0, 1 , 1, 1, 1] -> [ 0 , 1 ];
  [ 0 , 0, 0 , 0, 0, 1] -> [ 0 , 0 ];

end regfb

```

Figure D-6**No Report Simulation Output**

```

Simulate SYNARIO 1.00  Date Tue Aug 14 10:54:16 1993

Fuse file: 'regfb.jed'  Vector file: 'regfb.tmv'  Part: 'PLA'
Operation of the simulator on devices with feedback
        Data I/O Corp.      31 July 1993

4 out of 4 vectors passed.

```


In Figure D-7, one of the test vectors was changed to produce an error, and simulation was run through JEDEC simulation at Report Type: None (only errors are shown). When an error occurs, the simulation output lists the number of the vector that failed, the name and number of the failed output, and the nature of the failure.

Figure D-7**No Report Simulation Output Showing Error**

```
Simulate SYNARIO 1.00  Date Mon Aug  6 14:48:31 1993

Fuse file: 'fb2.jed'  Vector file: 'fb2.jed'  Part: 'P16R4'

SYNARIO 1.00 Data I/O Corp. JEDEC file for: P16R4 V8.0
Created on: Mon Aug  6 14:45:51 1993

Operation of the simulator on devices with feedback
      DATA I/O Corp.      31 July 1993

Vector 4
F2 13, 'L' found  'H' expected

3 out of 4 vectors passed.
```

Report Type: Tabular

Report Type: Tabular is the default format. Tabular gives a table with signal levels represented by H, L, and Z for logic high, logic low, and high-impedance state.

Figures D-2, D-3, and D-4 earlier in this appendix show the Tabular output for all Trace Types.

Report Type: Pins

The Pins output shows the actual signal outputs and the test vectors used to perform the simulation. The actual output associated with each test vector is shown on one line followed by the input portion of the test vector, Vector In, on the next line. The output portion of the test vector; that is, the expected output of the device appears on the Vector Out line below the actual output. Figure D-8 shows an example of a Pins report.

Figure D-8**Pins Report Simulation Output**

```
Simulate SYNARIO 1.00X Date Wed Aug 29 10:21:15 1993
Fuse file: 'fb2.jed' Vector file: 'regfb.tmv' Part: 'P16R4'
SYNARIO 1.00X Data I/O Corp. JEDEC file for: P16R4 V8.0

Created on: Wed Aug 29 10:21:07 1993
Operation of the simulator on devices with feedback

Vector 1
Vector In [C0111.....0.....]

Device In [001110000000011110001111]
Device Out [.....ZLHHHHZZ.....]

...edited...

Vector 4
Vector In [00001.....0.....]

Device In [000010000000001110000111]
Device Out [.....ZLLHHHZZ.....]

4 out of 4 vectors passed.
```

Report Type: Macro-cell

Macro-cell reports provide the same information as Tabular, plus internal device information such as OR-gate outputs, register outputs, and the final outputs. Figure D-9 shows one portion of a macro listing with pointers to its various parts and a portion of the corresponding logic diagram. Only a portion of the macro simulation output is shown since macro output files can be quite large.

Figure D-9
Macro-cell Report Simulation Output

```

Simulate SYNARIO 1.00X  Date Wed Aug 29 10:22:32 1993

Fuse file: 'fb2.jed' Vector file: 'regfb.tmv' Part: 'P16R4'

SYNARIO 1.00X Data I/O Corp. JEDEC file for: P16R4 V8.0
Created on: Wed Aug 29 10:21:07 1993

Operation of the simulator on devices with feedback
      Data I/O Corp.      31 July 1993

Vector 1
Vector In [C0111.....0.....]

Pin 11 [0
        ]-----
           F1 Pin 14  |  \
                   |  \>O--- H   Vec=H
                   |  /
                   |  /
           |  Q = L  |  --
           |  OR = L  |
           |  CK = L  |
           |-----

PT 1280 [FFFFFFF
Pin 1 [0
      ]-----

           F2 Pin 13  |  \
                   |  \>O--- L   Vec=L
                   |  /
                   |  /
           |  OR = H  |
           |-----

PT 1536 [T
Pin 1 [0
      ]-----

           F2 Pin 13  |  \
                   |  \>O--- L   Vec=L
                   |  /
                   |  /
           |  OR = H  |
           |-----

PT 1568 [TFFFFFFF
Pin 1 [0
      ]-----

Vector Out [.....LH.....]
...

```

Fuse and node numbers shown on the table are numbers assigned by Synario to the fuses in the device and are shown in the Logic Diagrams provided with the PLD Device Kit. The OR-gate and register outputs shown in the simulation output are internal signals not available as pin outputs that can be useful for debugging designs.

The Macro-cell report option produces large files if all pins and nodes are traced for all vectors. To get a reasonably-sized file, use the Watch Signals and Vector Range to Display properties to specify the pins or nodes for only the desired vectors. If you do not specify which signals to watch, the first I/O pin in the device is traced.

Table D-1 defines the notation used in the simulation macro output files to identify product terms and nodes.

Table D-1
Notation Used in Simulation Macro-cell Report Files

Notation	Description
Current Nodes	
OE	Output enable
AR	Asynchronous Reset
SR	Synchronous Reset
AP	Asynchronous Preset
SP	Synchronous Preset
LD	Register Load
CK	Register Clock
OR	Normal output OR gate ("D" "T")
IN1	First input to a Flip/Flop ("J" "S")
IN2	Second input to a Flip/Flop ("K" "R")
OR Node Types	
PTnnnn	One or more product term
LOW	Always logic level 0
HIGH	Always logic level 1
Pin nn	Input from pin
Node nn	Input or feedback from a internal node
Pin nn & nn	The AND of two pins
Pin nn # nn	The OR of two pins

Notation	Description
PROM nn	Bit nn of a prom output
<i>PRODUCT Term Display</i>	
Pin nn [1]	Logic level 1 from a pin or node
Pin nn [0]	Logic level 0 from a pin or node
nn & nn [0 & 1]	Logic level 0 from a pin or node
PTnnnnn [TTFFTT]	Multiple product terms
PTnnnnn [TT \$ FT]	XOR of two groups of product terms
PTnnnnn [TT # FT]	OR of two groups of product terms
PTnnnnn [T-FF-T]	Shared product terms ('-' term not connected)
PTnnnnn [TTTTTFTFTTTTFFFTTF] [TTTTTTTTTF]	Multiple line display of large OR
T = logic true; F = logic false	

Report Type: Wave

Report Type: Wave provides a graphic representation of the inputs and/or outputs of the device for each of the specified test vectors. The Vector Range to Display property (break points) specifies the vectors to be used and the Watch Signals property specifies which inputs and outputs appear in the output file. Up to 14 pins can be specified (blank columns can be inserted by entering 999, but they count as a pin in the output). If you do not specify which signals to watch, JEDEC simulation automatically generates the signals appearing at the first 14 output pins.

The JEDEC simulator watches only those signals specified in the test vectors.

Simulation and Designs with Buffered Outputs

When a design with 3-state buffered outputs is simulated with wave and/or tabular reports, the states of the outputs are reported as H, L, 1, 0, Z, or X, depending on the test vectors used, whether the pin is bidirectional, and whether the output buffer is enabled.

With a Tabular report, device pins that are output-only, or are bidirectional and configured as outputs, the outputs are reported as follows (in order of significance):

- ◆ If the buffer is enabled, the active state (H or L) of the output (that results from the levels applied at the input pins by the input test vector) is reported.
- ◆ If the buffer is not enabled, the same value (1, 0, Z, or X) applied to that output by the input test vector is reported.
- ◆ If the output is not enabled and no 1 or 0 is applied to that output by the input test vector, Z is reported.

Simulation and Unspecified Inputs

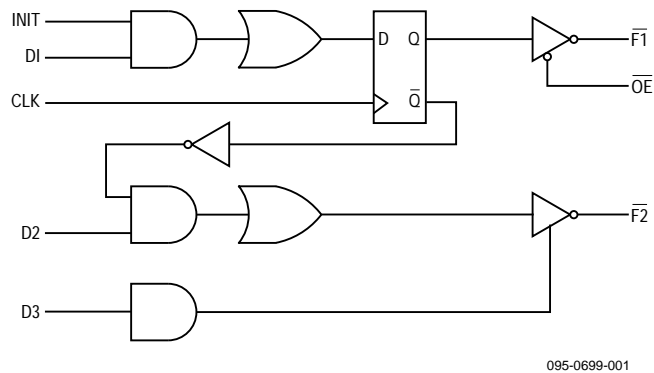
When the input test vector does not specify a logic level to be applied to a particular input, or set of inputs, simulation uses the default value assigned by the -x option. Using don't cares (Xs) in the input test vector can cause the input(s) to be unspecified. The JEDEC simulator does not propagate unknown values.

Simulation for Designs with Feedback

Logic designs containing feedback present a unique simulation problem, because the current output on one or more gates depends on the outputs of other gates. Thus, determining the outputs of a design with feedback is not a simple input-to-output determination. Propagation delays, the number of gates in the feedback path, and, in synchronous feedback circuits, clock inputs must be taken into account. When an input to the design changes, the outputs may not stabilize immediately. Synchronous circuits must be clocked before the outputs reflect changes in the inputs.

JEDEC simulation determines the final outputs of feedback circuits through iteration, calculating, and monitoring the outputs until they stabilize or are clocked out to give the final outputs. If outputs do not stabilize after 20 iterations, an error message is given. The iterations, final outputs, and the states of the internal register are provided in the simulation output file depending on the report and trace type you choose. Figure D-10 shows a simple synchronous circuit with feedback. One clock pulse is required after the inputs change to cause a corresponding change in the outputs. The source file describing this circuit and the simulation output for a Tabular report are shown in Figures D-5 through D-4.

Figure D-10
Synchronous Feedback Circuit

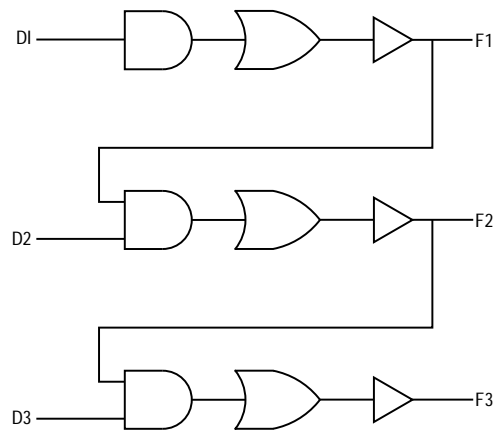


The Tabular report shows the test vectors and the final outputs after the clock pulse. A tabular report with clock trace shows the test vectors and the value of the outputs before and after the clock. A macro-cell report results in a large simulation output file. If you wish to examine the trace output for this circuit, you can run JEDEC simulation on regfb.abl with macro-cell report and examine the simulation results.

The second feedback example in Figure D-11 shows an asynchronous circuit that requires more than one simulation iteration before the outputs stabilize. Figure D-12 shows the source file describing the circuit and Figures D-13 and D-14 show the simulation output for brief trace and detailed trace.

Brief trace shows the final outputs after they have stabilized, as well as the test vectors. Detailed trace shows the output values at the different iterations as the outputs stabilize, as well as the final outputs and the test vectors. Notice that for the inputs provided in vector 2, three iterations are needed before the outputs stabilize. Vector 1 requires only one iteration to provide stable outputs. Macrocell report is not shown but can be generated by running JEDEC simulation with "feedback.abl" shown in Figure D-12.

Figure D-11
Asynchronous Feedback Circuit



095-0700-001

Figure D-12
Source File: Asynchronous Feedback Circuit

```

module feedback
  title 'Operation of the simulator on devices with feedback'
  FB1    device 'P16HD8';
  D1,D2,D3    pin 1,2,3;
  F1,F2,F3    pin 13,14,15;
equations
  F1    = D1;
  F2    = D2 & F1;
  F3    = D3 & F2;
test_vectors ([D1,D2,D3] -> [F1,F2,F3])
  [ 0, 0, 0] -> [ 0, 0, 0];
  [ 1, 1, 1] -> [ 1, 1, 1];
end feedback

```


Figure D-13

Brief Trace Simulation Output: Asynchronous Feedback Circuit

```
Simulate SYNARIO 1.00 Date Mon Aug 6 14:55:57 1993
Fuse file: 'feedback.jed' Vector file: 'feedback.tmv'
Operation of the simulator on devices with feedback
DATA I/O Corp. 24 Feb 1993
      D D D   F F F
      1 2 3   1 2 3
V0001 0 0 0   L L L
V0002 1 1 1   H H H
```

Figure D-14

Detailed Trace Simulation Output: Asynchronous Feedback Circuit

```
2 out of 2 vectors passed.
Simulate SYNARIO 1.00 Date Mon Aug 6 14:57:01 1993
Fuse file: 'feedback.jed' Vector file: 'feedback.tmv'
Operation of the simulator on devices with feedback
DATA I/O Corp. 24 Feb 1993
      D D D   F F F
      1 2 3   1 2 3
V0001 0 0 0   L L L
V0002 1 1 1   H L L
      1 1 1   H H L
      1 1 1   H H H
2 out of 2 vectors passed.
```

Register Preloads in the Simulator

When using a preload vector as the first test vector in the simulator, and the device being used has asynchronous presets, you should include a "dummy" vector before the preload vector (that is, all "don't cares"). The dummy vector prevents the possibility of the simulator initialization destroying the preload data. Care should be taken when setting don't cares to 1; when preloading registers to the 1 state, be sure to define values for the resets of the preloaded registers. If the resets are not defined, the registers will be reset after the preload operation, and the preloaded data will be lost.

Test Vectors and Simulation

JEDEC simulation simulates the programmed device with user-supplied design test vectors. The more comprehensive and detailed your test vectors are, the more useful your simulation results will be.

With test vectors, you specify the required input pattern and expected outputs at the device pins. JEDEC simulation will apply the inputs from the test vectors to the simulated circuit and compare the simulated output with the output specified in the test vectors. If there is any difference, an error is indicated.

Note that the simulators cannot test a mode in your design if you do not write a test vector to force that mode of operation. It is to your advantage to create complete sets of test vectors that test all functions of your logic design, and to use JEDEC simulation regularly as design changes are made.

***Note** The nodes shown on logic diagrams (shipped with the PLD Device Kit) are outputs for writing equations; for example, the OR terms for the RS flip/flops in an F167. These nodes cannot be used as direct flip/flop inputs for simulation test vectors.*

Debugging State Machines

State machines can be difficult to debug once an error occurs because each state depends on previous states and affects subsequent states. An error in the description of one state transition can cause the state machine to follow a different sequence from the test vectors. The loss of synchronization can cause a cascade of errors that makes the original error hard to isolate.

To test and debug larger state machines:

- ◆ Add test vectors that periodically force the machine to known states to reset the state machine and eliminate cascading of errors.
- ◆ Write small sets of test vectors that test individual functions of the state machine, and gradually add them to the simulation.

Periodically forcing (with your test vectors) the state machine to a known state and then letting the state transitions take place limits the cascading of errors to a smaller number of states and makes it much easier to find initial errors.

Starting with a small set of test vectors that tests only part of the state machine's function also helps to isolate any errors. When operation of one function has been verified, add a set of test vectors that test another function, then add another, and so on, until you have tested the full function of the state machine. Combining this technique of gradual simulation with the forcing vectors discussed above makes errors easier to pinpoint and simplifies the testing of large state machines.

Multiple Test Vector Sections

More than one set of test vectors can be used to simulate the function of a device. This may be useful in the representation of the test in the source file. In a similar manner, any time you have two or more distinct functions being performed by the same device, you may want to describe the vectors in separate sections for each function.

Take for example, the source file presented in Figure D-15 that describes AND and NAND gates implemented on the same device as well as the test vectors used to simulate the operation of that device. The test vectors are described in two separate sections. The first test vectors section lists the test vectors that simulate the operation of the AND portion of the design, and the second section tests the NAND function. With the test vectors written as they are, in two separate sections, the correspondence between test vectors and the function being tested is readily apparent in the source file.

Figure D-15
Source File with Multiple Test Vector Sections

```

module simple
  title 'Simple ABEL-HDL example Dan Poole Data I/O Corp'
      U7      device 'P14H4';
      A1,A2,A3  pin 1,2,3;
      N1,N2,N3  pin 4,5,6;
      AND,NAND  pin 14,15;
equations
  AND      = A1 & A2 & A3;
  !NAND    = N1 & N2 & N3;
test_vectors 'Test And Gate'
  ( [A1,A2,A3] -> AND )
  [ 0, 0, 0] -> 0;
  [ 1, 0, 0] -> 0;
  [ 0, 1, 0] -> 0;
  [ 0, 0, 1] -> 0;
  [ 1, 1, 1] -> 1;
test_vectors 'Test Nand Gate'
  ( [N1,N2,N3] -> NAND )
  [ 0, 0, 0] -> 1;
  [ 1, 0, 0] -> 1;
  [ 0, 1, 0] -> 1;
  [ 0, 0, 1] -> 1;
  [ 1, 1, 1] -> 0;
end simple

```

Automatic Signal Selection

The JEDEC simulator shows only signals used in test vectors for all report formats (except for pins), unless you specify specific signals with the Watch Signal property. (The Pins report type shows all inputs and outputs.) If you run the module shown above (module simple), with the default tabular format, the simulation results are given for only the signals used in the test vectors (Figure D-16):

Figure D-16
Simulation Results Showing Automatic Signal Selection

```
Simulate SYNARIO 1.00  Date Thu Aug  9 14:45:02 1993

Fuse file: 'simple.jed'  Vector file: 'simple.tmv'

Simple ABEL-HDL example

***** Test And Gate *****

          A
        A A A  N
        1 2 3  D

V0001  0 0 0  L
V0002  1 0 0  L
V0003  0 1 0  L
V0004  0 0 1  L
V0005  1 1 1  H

***** Test Nand Gate *****

          N
          A
        N N N  N
        1 2 3  D

V0006  0 0 0  H
V0007  1 0 0  H
V0008  0 1 0  H
V0009  0 0 1  H
V0010  1 1 1  L
10 out of 10 vectors passed.
```

Don't Cares in Simulation

In ABEL-HDL, you can use the special constant *.X.* in a test vector to denote a don't-care input or output. The *.X.* tells the JEDEC simulator to choose a value for the input designated by the *.X.* in the test vector(s), or to disregard an output signal's state. The default value used in the simulator for the don't care inputs is zero. However, you can use the Don't Care (X) Value property of Simulate JEDEC File to specify zero or one (0 or 1) for the don't care value.

Input pins that are not specified in the test vectors are given the default don't care value zero (0) by the simulator unless the don't care property is set to 1. In this case, all unspecified pins will be assigned a value of 1 in the test vectors.

If you experience trouble with devices not working in a circuit or programmer/tester, it may be helpful to recheck the don't care assumptions. There may be a combination of 1s and 0s in a test vector that needs to be checked by the JEDEC simulator.

Note *Logic programmers use JEDEC vectors to test the device. Occasionally vectors that pass in JEDEC simulation will fail on the programmer. Data I/O logic programmers such as the UniSite, 2900 or 3900 show on the operator's terminal the pin and vector that failed.*

Also, the simulator checks the design with a single level for the don't care inputs, while the target circuit may place other levels on the input during actual operation of the device. For complete simulation, you must run the JEDEC simulation operation with the don't cares set to 0, and then again with them set to 1.

The simulators ignore output pins that are not specified in the test vectors and will not indicate an error due to conflict between a specified value and the value determined by the simulator. The .X. constant at an output pin tells the simulators not to compare the outputs (the output produced by the design and the output specified in the test vector) but still allow them to be displayed. Figure D-17 shows how the .X. value can be assigned to the outputs prior to the JEDEC simulation step of the language processor.

Figure D-17

Assignment of Don't Care Value (.x.) to Design Outputs

```
module findout
title 'The JEDEC simulator will find the output levels
Ngoc Nicholas    Data I/O Corp    9 Aug 1993'

    F1      device 'P16L8';
    A,B,Y1,Y2    pin 1,2,14,15;
    X = .X.;

equations
    !Y1 = A # B;
    !Y2 = A $ B;

test_vectors
    ([A,B] -> [Y1,Y2])
    [0,0] -> [ X, X];
    [0,1] -> [ X, X];
    [1,0] -> [ X, X];
    [1,1] -> [ X, X];
end
```

Using Pins, Wave and Tabular report types, you can observe the actual output values determined by the simulator. In Figure D-18, the X entries (in the test vectors) for pins 14 and 15 allow the simulator to display an H or L to indicate the output value for the specified inputs.

Figure D-18

JEDEC Simulation Results with Outputs Specified as Don't Care

```
Simulate SYNARIO 1.00X    Date Tue Aug 28 15:07:54 1993

Fuse file: 'findout.jed' Vector file: 'findout.tmv'

The JEDEC simulator will find the output levels
Ngoc Nicholas    Data I/O Corp    9 Aug 1993

    Y Y
    A B  1 2

V0001  0 0   H H
V0002  0 1   L L
V0003  1 0   L L
V0004  1 1   L H
4 out of 4 vectors passed.
```

Preset and Preload Registers

Preset, reset, and preload are terms used to define a specific action and resultant output of one or more registers contained in a programmable logic device. Preset forces all register outputs to one, reset forces all register outputs to zero, and preload forces all registers to specified states. Synchronous preset, reset, and preload functions require a clock input. Asynchronous functions require no clock input.

To verify the operation of these devices, appropriate test vectors must be written and placed in the source file. These test vectors allow the JEDEC simulation steps of the language processor to verify operation of the design by performing the required operations of these registers.

Note For preload, the JEDEC Simulator assumes that devices have inversion between the register outputs and the device outputs. When preloading devices that have noninverting outputs or that have outputs programmable to noninverting, the data to be preloaded must be complemented to obtain the desired preload condition.

Also note that it is not possible to preset, reset, and preload at the same time. Preset and preload must not contend during preload with other inputs, preset, or register functions.

Special Preset Considerations

Certain programmable logic devices, such as the F105 and F167, do not respond to the first clock pulse following a preset condition (power-on or the preset input). These devices allow normal clocking only after a high-to-low transition of the clock input following the preset condition. Therefore, simulation for these devices requires an additional test vector following the preset condition to provide the high-to-low transition of the clock input.

To illustrate the preset considerations for these devices, a four-state counter with clock and preset inputs is presented in Figure D-19, along with the test vectors required to properly verify the design. The equation for the preset condition is written using the dot extension for the two registers. This counter is targeted for a circuit that provides a power-on preset condition; so the test vectors must verify operation of the counter after power-on preset as well as after the preset input has been active.

Figure D-19**Test Vectors for Special Preset Conditions**

```

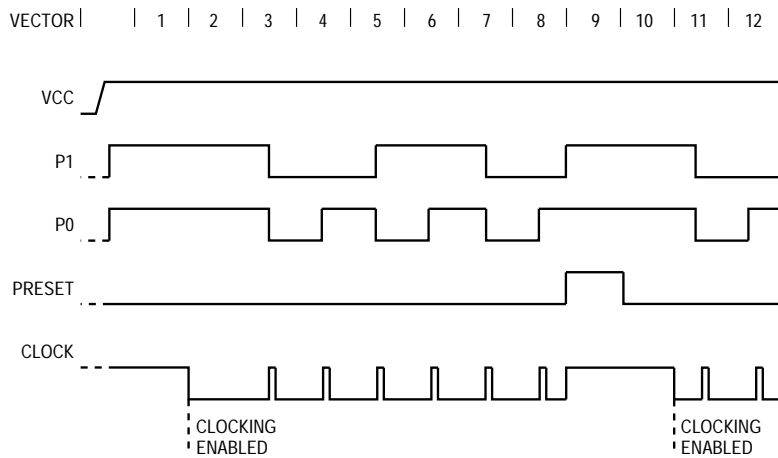
module preset
  title '2-bit counter to demonstrate power on preset Bob Hamilton  Data I/O Corp'
  preset device 'F167';
  Clk, Hold      pin 1,2;
  PR             pin 16;          "Preset/Enable
  P1,P0         pin 15,14;
  Ck,X = .C,..X.;
equations
  [P1,P0].PR = PR;
  [P1,P0].C = Clk;
  [P1.R,P0.S] = !P1 & !P0 & !Hold;  " state 0
  [P1.S,P0.R] = !P1 & P0 & !Hold;   " state 1
  [P1.S,P0.S] = P1 & !P0 & !Hold;   " state 2
  [P1.R,P0.R] = P1 & P0 & !Hold;    " state 3
test_vectors
  ([Clk,PR,Hold] -> [P1,P0])
  [ 1 , 1, 0 ] -> 3;
  [ 1 , 0, 0 ] -> 3; " Provides a High-to-Low on clock
  [ 0 , 0, 0 ] -> 3; " to enable clocking
  [ Ck, 0, 0 ] -> 0;
  [ Ck, 0, 0 ] -> 1;
  [ Ck, 0, 0 ] -> 2; " Hold count
  [ Ck, 0, 1 ] -> 2;
  [ Ck, 0, 0 ] -> 3;
  [ Ck, 0, 0 ] -> 0; " Roll over
  [ Ck, 0, 0 ] -> 1;
  [ 1 , 1, 0 ] -> 3; " Preset high
  [ 1 , 0, 0 ] -> 3; " Preset low
  [ Ck, 0, 0 ] -> 0;
  [ Ck, 0, 0 ] -> 1;
" Notes edited...
end

```

Figure D-20 is a timing diagram that shows the action of the test vectors in Figure D-19. As indicated in the timing diagram, the preset input overrides the clock input and when held high, inhibits clocking of the counter. Assuming that the device is powered up in the preset condition, the first test vector pulls the clock input high while the second vector pulls it low to provide the transition required for normal clocking.

The next six test vectors provide clock inputs to increment the counter through all states and back to state one. As shown in the timing diagram, the 9th test vector invokes the preset function, and the 10th test vector pulls the preset input low and maintains the clock input high. The 10th test vector allows the preset line to go low before the high-to-low transition of the clock. The preset line must go low before the clock so that the high-to-low clock transition can enable the clock pulse of the 11th test vector. The high-to-low transition that follows the 10th test vector resumes normal clocking of the device.

Figure D-20
Timing Diagram Showing Test Vector Action



Preset overrides clock, and when held high, clocking is inhibited and the registers are high.
Normal clocking resumes with the first clock pulse following a high-to-low clock transition after preset goes low.

095-0748-001

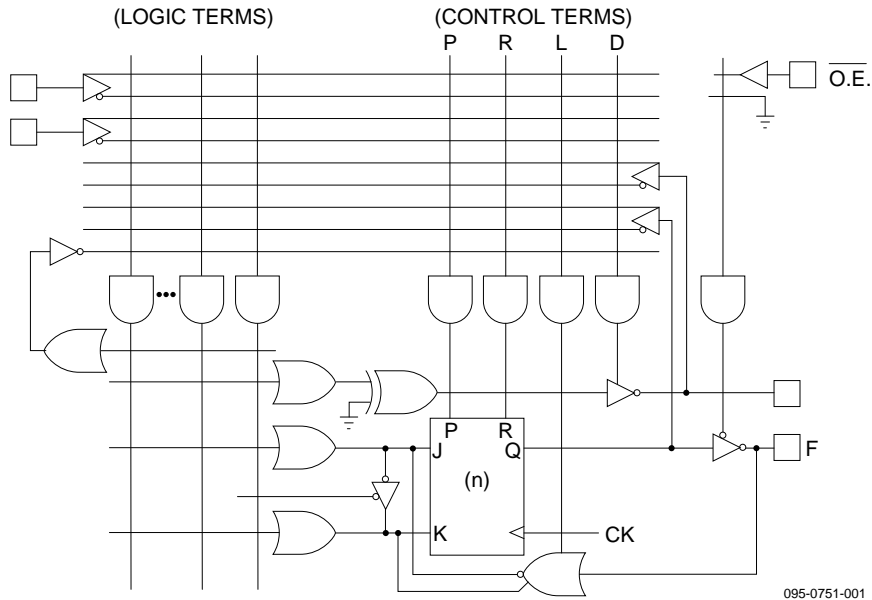
If the 2nd and 10th test vectors were not included in the source file, the clock pulse of the 3rd and 11th vectors would be lost. That is, the high-to-low transition of the clock pulses in these vectors would cause the resumption of normal clocking, but would not increment the counter as required by the design.

TTL Preload

Certain programmable logic devices, such as the F159 FPLS (field programmable logic sequencer) allow internal registers to be forced (preloaded) to a known state by means of the TTL preload function. Figure D-21 shows a typical FPLS layout. To preload the output register in such a device, four conditions must be present:

- ◆ The output is placed in the high-impedance state
- ◆ The desired register state is placed on the output pin
- ◆ The load control term is activated
- ◆ A clock pulse is applied to the clock input

Figure D-21
Internal Register of the F159



The fifth test vector in Figure D-22 shows how each of the above conditions are met. In the fifth test vector

- ◆ OE (Ena) pin is held at 1
- ◆ The output (F0) is pulled low by inserting a 0 in the input side of the test vector
- ◆ A 1 is applied to the LOAD input, which activates the load control term (LA)
- ◆ A clock pulse is provided by the C (.C.) applied to the clock input

Figure D-22
Invoking the TTL Preload Function

```

module TTLload
  title 'TTL load example
  Dave Kohlmeier   Data I/O Corp  9 Aug 1993'

  TTL59  device  'F159';

  C,L,H,X,Z      = .C.,0,1,.X.,.Z.;

  Clk,J_IN,K_IN,LOAD,Ena,F0      pin 1,2,3,4,11,12;

  F0 istype 'reg_JK,invert';

equations
  F0.OE  = !Ena;
  F0.J   = J_IN;
  F0.K   = K_IN;
  F0.L   = LOAD;
  F0.C   = Clk;

test_vectors
  ([Clk,Ena,J_IN,K_IN,LOAD,F0] -> F0)
  [ C , L , 1 , 0 , 0 , X] -> 0 ; "Set
  [ C , L , 0 , 1 , 0 , X] -> 1 ; "Reset
  [ C , L , 1 , 1 , 0 , X] -> 0 ; "Toggle
  [ C , L , 1 , 1 , 0 , X] -> 1 ; "Toggle
  [ C , H , 0 , 0 , 1 , 0] -> X ; "Load
  [ 0 , L , 0 , 0 , 0 , X] -> 0 ; "Test
  [ C , L , 1 , 1 , 0 , X] -> 1 ; "Toggle
end

```

The sixth test vector tests to make sure the register was loaded with the 0 applied to the output by the fifth test vector. The sixth test vector enables the output (Ena at 0) and allows the output to be tested (F0 = X on the input side and 0 on the output side of the vector) while holding the clock input at 0.

Supervoltage Preload

Supervoltage preload allows the setting of registers within certain devices, such as the P16R4, to the logic levels placed on their registered outputs. Supervoltage preload is accomplished by means of the .P. test condition (.P. special constant) that is used to "jam load" registers within the logic device to the desired state. When the .P. test condition is applied to the clock pin, the logic level applied to the register output is loaded into the register. Devices with separate banks of registers require that the P test condition be applied to each clock pin. Also during preload, certain device pins, such as the output enable pin, may have to be in a defined state.

To verify the preload operation, use a separate test vector to test the outputs. This vector must follow the vectors that perform the preload operation.

Supervoltage preload can be used to test state machine designs that could assume one or more illegal states, or designs that contain branch conditions. An illegal state for a state machine is a state that the design does not allow, but the device is capable of assuming under certain conditions (such as powerup or noise). A typical decade counter having states 0 through 9 and four registers, could possibly assume six additional (and illegal) states (10 through 15). The decade counter should be designed so that when an illegal state is reached, the next clock pulse returns the counter to the 0 state. During simulation it is necessary to not only test the counter for normal up/down/clear operation (performed by the test vectors) but also to insure that it will clock to state S0 from any illegal state.

To test that a decade counter will clock to state 0 from any illegal state, it is necessary to do two things:

1. Define all illegal states to be tested.
2. Create test vectors that verify the return to state 0 from any illegal state.

To test the illegal states for a decade counter, it is necessary to define the illegal states (that is, 10 through 15). Figure D-23 shows that the following additional entries are included to define the six illegal states:

```
S10= ^b0101;  
S11= ^b0100;  
S12= ^b0011;  
S13= ^b0010;  
S14= ^b0001;  
S15= ^b0000;
```

To verify that the design will recover from each of the illegal states, appropriate test vectors are included. This group of test vectors preloads the device to each possible illegal state and then verifies that the device clocks to state S0. The test vectors preload the counter by means of the .P. special constant applied to the clock pin and a logic high applied to the output enable (OE) pin. The test vector that follows the preload test vector verifies the result of the preload operation, while the following test vector verifies the clocking of the counter from the illegal state to state S0.

Figure D-23**Defining Illegal States and Test Vectors for Illegal States**

```

module CNT10P
  title 'decimal counter
Note: preload the data on pins into the registers, Denny Siu   Data I/O Corp'

  cnt10p          device 'P16R4';

  Clk,Clr,OE      pin 1,2,11;
  Q3,Q2,Q1,Q0     pin 14,15,16,17 istype 'reg_D,invert';

  Ck,X,Z,P        = .C. , .X., .Z., .P.;

" Counter States
  S0 = ^b1111;    S4 = ^b1011;    S8 = ^b0111;    S12= ^b0011;
  S1 = ^b1110;    S5 = ^b1010;    S9 = ^b0110;    S13= ^b0010;
  S2 = ^b1101;    S6 = ^b1001;    S10= ^b0101;   S14= ^b0001;
  S3 = ^b1100;    S7 = ^b1000;    S11= ^b0100;   S15= ^b0000;

equations
  [Q3,Q2,Q1,Q0].c = Clk;
  [Q3,Q2,Q1,Q0].oe = !OE;

state_diagram [Q3,Q2,Q1,Q0]
  State S0:      IF !Clr THEN S1 ELSE S0;
  State S1:      IF !Clr THEN S2 ELSE S0;
  State S2:      IF !Clr THEN S3 ELSE S0;
  State S3:      IF !Clr THEN S4 ELSE S0;
  State S4:      IF !Clr THEN S5 ELSE S0;
  State S5:      IF !Clr THEN S6 ELSE S0;
  State S6:      IF !Clr THEN S7 ELSE S0;
  State S7:      IF !Clr THEN S8 ELSE S0;
  State S8:      IF !Clr THEN S9 ELSE S0;
  State S9:      GOTO S0;

"Ensure return from illegal state
  State S10:     GOTO S0;
  State S11:     GOTO S0;
  State S12:     GOTO S0;
  State S13:     GOTO S0;
  State S14:     GOTO S0;
  State S15:     GOTO S0;

```

```

@page
test_vectors 'Test Counter'
  ( [Clk ,OE, Clr ] -> [Q3,Q2,Q1,Q0])
  [ Ck , 0, 1 ] -> S0;
  [ Ck , 0, 0 ] -> S1;
  [ Ck , 0, 0 ] -> S2;
  [ Ck , 0, 0 ] -> S3;
  [ Ck , 0, 0 ] -> S4;
  [ Ck , 0, 0 ] -> S5;
  [ Ck , 1, 0 ] -> Z ;
  [ Ck , 0, 0 ] -> S7;
  [ 0 , 0, 0 ] -> S7;
  [ Ck , 0, 0 ] -> S8;
  [ Ck , 0, 0 ] -> S9;
  [ Ck , 0, 0 ] -> S0;
  [ Ck , 0, 0 ] -> S1;
  [ Ck , 0, 0 ] -> S2;
  [ Ck , 0, 1 ] -> S0;

test_vectors 'preload to illegal states'
  ( [Clk ,OE, Clr,[Q3,Q2,Q1,Q0]] -> [Q3,Q2,Q1,Q0])
  [ P , 1, 0 , S10 ] -> X ;
  [ 0 , 0, 0 , X ] -> S10;
  [ Ck , 0, 0 , X ] -> S0 ;
  [ P , 1, 0 , S11 ] -> X ;
  [ 0 , 0, 0 , X ] -> S11;
  [ Ck , 0, 0 , X ] -> S0 ;
  [ P , 1, 0 , S12 ] -> X ;
  [ 0 , 0, 0 , X ] -> S12;
  [ Ck , 0, 0 , X ] -> S0 ;
  [ P , 1, 0 , S13 ] -> X ;
  [ 0 , 0, 0 , X ] -> S13;
  [ Ck , 0, 0 , X ] -> S0 ;
  [ P , 1, 0 , S14 ] -> X ;
  [ 0 , 0, 0 , X ] -> S14;
  [ Ck , 0, 0 , X ] -> S0 ;
  [ P , 1, 0 , S15 ] -> X ;
  [ 0 , 0, 0 , X ] -> S15;
  [ Ck , 0, 0 , X ] -> S0 ;
end

```

An example of a state machine that contains branch conditions is given in the blackjack machine in the *Synario ABEL-HDL Reference*. If it was not possible to preload the state machine to each branch condition, it would be necessary to repeat the test vectors down to the Test22 state for each branch of the Test22 state. The test vectors in Figure D-24 show how this state machine can be preloaded to test these three branches of the design.

Figure D-24
Using Test Vectors to Preload a State Machine

```

test_vectors ' Test 3 way branch at Test22'
([Ena,Clk,LT22,Ace,Qstate] -> [Ace,Qstate ])
[ 1 ,.P., X 1 ,Test22] -> [ X, X ];
[ 0 , O , 1 X , X ] -> [ H,Test22 ]; "Verify preload
[ 0 , C , 1 X , X ] -> [ H,ShowStand];
[ 1 ,.P., X 0 ,Test22] -> [ X, X ];
[ 0 , O , 0 X , X ] -> [ L,Test22 ];
[ 0 , C , 0 X , X ] -> [ L,ShowStand];
[ 1 ,.P., X 1 ,Test22] -> [ X, X ];
[ 0 , C , 0 X , X ] -> [ H,Sub_10 ];

```

Preset/Reset Controlled by Product Term

In programmable logic devices such as the P22V10, preset and reset functions are controlled by product terms. An example of controlling the preset and reset functions is given in Figure D-25.

Figure D-25
Controlling Reset/Preset by Product Term

```

module reset22a
title 'Demonstrates Asynchronous Reset and Synchronous Preset'
    reset22a      device 'P22V10';
    Clk,I1,I2,R,S,T Pin 1,2,3,4,5,6;
    Q1,Q2        Pin 14,15 istype 'buffer';
    Ck,Z,H,L     = .C., .Z., 1, 0;
    Input        = [I2,I1];
    Output       = [Q2,Q1];

equations
    Output      := Input;    "Registered buffer
    Output.AR   = R & !T;
    Output.SP   = S & !T;
    Output.Clk = Clk;

test_vectors
    ([Clk,Input,R,S,T] -> Output)
    [ Ck, 0 ,0,0,0] -> 0;
    [ Ck, 1 ,0,0,0] -> 1;
    [ Ck, 2 ,0,0,1] -> 2;
    [ 0 , 3 ,0,0,1] -> 2;    "Hold
    [ Ck, 3 ,0,0,1] -> 3;
    [ 0 , 3 ,1,0,1] -> 3;    "Reset = R & !T
    [ 0 , 3 ,1,0,0] -> 0;    "Async Reset
    [ 0 , 0 ,0,1,0] -> 0;    "Preset requires clock
    [ Ck, 0 ,0,1,0] -> 3;    "Sync Preset

end

```

Note In devices like the P22V10, there is a common reset for all registers. While a reset equation for one output will reset all of the registers in the P22V10, you should write individual reset equations to ensure architecture-independence.

Note For devices that have programmable polarity at the output of the registers, preset and reset functions may complement the output pins.

In the P22V10, the reset is asynchronous while preset is synchronous (to the clock input). The test vectors in Figure D-25 verify the reset and preset functions. The first three vectors verify the loading of input data. (Note that the third vector pulls the T input high; this is in preparation for T to be pulled low later in the simulation.) The fourth vector verifies operation of the clock input by changing the input data without providing a clock input. The sixth vector verifies that the R input without a low T input will not provide the asynchronous reset.

The seventh vector verifies operation of the asynchronous reset by pulling the T input low. The next vector verifies that a high S input and low T input do not preset the device without the clock input. The final vector verifies the synchronous preset by providing the clock pulse and testing the output for both output pins at logic high (decimal 3).

Preset/Reset Controlled by Pin

Some devices have a direct Preset or Preload coming from a pin. Be sure to include this in the test vectors or simulation errors may occur.

Powerup States

Some devices power up with registers set to 1, some set to 0 and some set to an unknown value. For example, some TI devices power up with registers set and outputs low while some AMD devices power up with registers clear and outputs high. The first test vectors should always place the device in a known state. For example:

```
test_vectors
([clk,clear,input]->[output])
[ 0 , 0 , 0 ]->[ .X. ]; "Power up to don't care
test
[ 0 , 1 , 0 ]->[ 0 ]; "Clear the f/f and put
output to known value
[ 0 , 0 , 0 ]->[ 0 ]; "Relax clear signal and
test for steady state
[.C., 0 , 0 ]->[ 0 ]; "Begin clocking
```

Devices with Clock Inputs

Since devices with registered outputs must be clocked before the outputs reflect any change in inputs, a clock pulse must be specified as one of the test vector inputs. A clock input is indicated by a .C. in the test vector for a low-high-low pulse, and a .K. for a high-low-high pulse. The clock input causes JEDEC simulation to evaluate the inputs to the outputs prior to the first clock pulse transition (low-to-high or high-to-low depending on the polarity of the clock signal). The evaluation consists of the iterative steps described in "Simulation Program Operation." The inputs to outputs are then evaluated with the clock input at its active state, and then again with the clock input at its inactive state.

When running JEDEC simulation with Trace Type of Clock or Detailed, simulation data will be written for all three evaluations. That is, internal test vectors are generated to evaluate the design before the first clock transition, after the first clock transition, and after the second clock transition, thus effectively expanding the number of test vectors. An example of JEDEC simulation output for a device with a clock input is shown in Figure D-3 under "Trace Type: Clock" earlier in this appendix. The clock input is represented by the .C. on the Vector In line. On the four subsequent Device In lines, the .C. input goes from 0 to 1 and back to 0 to provide one complete clock pulse.

