

IspLEVER™ v1.0 Concepts Manual

Welcome to ispLEVER Concepts, a resource manual of design, implementation, verification, and programming concepts that comprise ispLEVER 1.0. This manual explains the purpose of each ispLEVER tool and gives guidelines for using these tools to implement Lattice programmable devices.

The contents of this document are also covered in ispLEVER Help. However, this document offers the benefit of being able to print multiple pages (topics) at a time.

Contents

<u>Overview of ispLEVER</u>	11
Constraint Editor	11
Hierarchy Browser	11
Hierarchy Navigator	11
ispEXPLORER	11
ispVM System	11
Lattice Logic Simulator	12
Library Manager	12
Performance Analyst	12
Project Navigator	12
Report Viewer	12
Schematic Editor	12
SVF Debugger	12
Symbol Editor	13
Text Editor	13
Waveform Editor	13
Waveform Viewer	13
CPLD Process Flow	14
Program Folder	14
Learning ispLEVER	15
ispLEVER Concepts	15
ispLEVER Help	15
ispLEVER Tutorial	15
Example Projects	15
Third-Party Documentation	16
Product Literature	16
Using ispUPDATE	17
Starting the ispUpdate Program	17
Using the Update Feature	18
Using the Download Feature	21
Launching the Patch Install Program	22
Migrating from ispDesignEXPERT 8.x (8.2, 8.3, or 8.4) to ispLEVER	23
MACH Devices	23
ispLSI Devices	23
Using ispLEVER Help	26
Context-sensitive Help	26
Running ispLEVER from the Command Line	26
Contacting Lattice Semiconductor	26
Headquarters	26
Product Support	27
<u>Project Management</u>	28
<u>Project Navigator</u>	28
The Project Navigator Interface	28
Valid Source Types	29
Source Hierarchy	29
Process Flows	30
Source-Level Flow	30

Project-Level Flow	30
Initialization	30
Using the Global Constraints Dialog Box to Control Optimization	31
Using the Location Assignments Dialog Box to Pre-assign Pins and Nodes	31
Describing a Project	31
Design Hierarchy	31
Tips for Defining Projects	31
Where Source Files are Placed	31
Processing a Design	32
Tip for Saving and Naming Projects	32
Forcing a Process to Run	32
Reserved File Names	33
Pop-up Menus	33
Hierarchical Design	34
What is a Hierarchical Design?	34
Advantages of Hierarchical Design	34
Hierarchy vs. Sheets	35
Approaches to Hierarchical Design	35
Hierarchical ABEL-HDL Design	36
Hierarchical Schematic Design	36
Hierarchical Verilog HDL Design	36
Hierarchical VHDL Design	36
Hierarchical Design Considerations	37
Hierarchical Design Structure	37
Hierarchical Naming	38
Nets in the Hierarchy	39
Automatic Aliasing of Nets	39
Hierarchical Design Examples	40
ABEL-HDL Hierarchy Example	40
Top-level ABEL-HDL Module (top.abl)	40
Lower-level ABEL-HDL module (add.abl)	41
Schematic Hierarchy Example	42
Lower-level ABEL-HDL Module for Add Block Symbol	43
Schematic/Verilog HDL Hierarchy Example	43
Lower-level Verilog HDL (mux2x1v.v)	44
Schematic/VHDL Hierarchy Example	45
Lower-level VHDL Module (mux2x1vhd.vhd)	46
Design Entry	47
ABEL-HDL Design	47
Using a Template to Create an ABEL-HDL Source	47
Entering Declarations	48
Entering Logic Descriptions	48
Entering Test Vectors	49
Verilog HDL Design	50
Adding a Verilog HDL Module to Your Design	50
Creating a New Verilog HDL Module	50
Synthesizing Your Verilog HDL Design	50
VHDL Design	52
Adding a VHDL Module to Your Design	52
Creating a New VHDL Module	52
Synthesizing your VHDL Design	53
EDIF Design	54
Importing an EDIF Netlist	54

Translating EDIF Properties	55
EDIF Properties	55
PIN LOCATION Property	55
GROUPING Property	56
OUTPUT SLEW Property	56
SIGNAL OPTIMIZATION Property	56
OPEN DRAIN Property	56
PULL Property	56
OUTPUT VOLTAGE Property	57
Schematic Design	58
Introduction to Schematic Design	58
Schematic Overview	58
What is a Schematic?	59
What do Schematics Consist of?	59
Symbols	60
Symbol Information	60
Graphics and Text	60
Pins	60
Attributes	61
Wires	61
Wire Names	61
Net Attributes	61
I/O Markers	62
Graphics	62
Text	62
Naming Schematic and Symbol Files	62
Schematic Attributes	63
Attribute Use	63
Attribute Types	63
Attribute Components	63
Attribute Name	63
Attribute Value	63
Attribute Modifier	63
Attribute Window	64
Setting Attribute Values	64
Default Values	64
Displaying Attribute Values on a Schematic	65
The Schematic Editor	66
Basic Schematic Editor Operation	66
Error Recovery	67
Network Operation	67
Wiring the Schematic	67
Nets	68
Net Names	68
Interconnection	68
Entering Net Names	68
Placing Net Names	69
Legal Characters in Net Names	69
Reserved Names	69
Logical Inversion	70
Specifying Signal Direction	70
Buses	70
Ordered Buses	70
Unordered Buses	71
Bus Taps	72

Naming the Tap	72
Connecting to Pins	72
Bus Pins	72
Nets on Iterated Instances	73
Compound Names	73
Single Names	73
Bus and Net Connections to Iterated Instance	73
Wiring Constraints	74
Debugging and Verifying a Schematic	75
"Unconnected Pin" Message	75
The Symbol Editor.....	76
Symbol Elements	76
Positioning Master Symbols	77
Using Grids to Position Symbol Elements	77
Positioning Pins	77
Selecting a Line Weight	77
Drawing Lines	77
Drawing Rectangles	78
Drawing Circles and Arcs	78
Drawing Negation Bubbles	78
Drawing Text	78
Text Size and Justification	78
Preparing Symbols for Schematics	78
Pins	78
Adding Pins	79
Bus Pins	79
Bus Pin Limitations	79
Attributes	80
Symbol Attributes	80
Pin Attributes	80
Attribute Windows	80
Setting Symbol Origins	81
Saving Symbols	81
Printing Symbols	81
Checking Symbols	81
The Hierarchy Navigator.....	83
Attributes	83
Attribute Modifiers	83
Attribute Window	85
Attribute Functions	85
Attribute Types	85
Attribute Names	86
Attribute Numbers	86
Attribute Values	86
The Library Manager.....	87
Why Use the Library Manager?	87
Working with Binary Symbol Libraries	87
The Hierarchy Browser.....	88
Introduction to the Hierarchy Browser	88
Mixed-Mode Design.....	89
The Text Editor.....	90

<u>Design Simulation</u>	91
Integrated Simulation	91
Standalone Simulation	91
Functional Simulation - CPLD Process Flow	92
<u>Lattice Logic Simulator</u>	93
Simulation Support	93
Stand-alone Simulation	93
Integrated Simulation	93
Simulation in the ispLEVER Design Flow	94
Design Entry	94
Test Stimulus	95
Creating Test Stimulus for Lattice Logic Simulator	95
Creating Graphic Waveforms	96
Creating Test Vectors	96
Creating a VHDL Test Bench	96
Creating a Verilog Test Bench	97
Special Constants	97
Using .P. to Pre-load Values in the State Machine	97
Viewing Simulation Results	99
Cross Probing	99
Simulation Mode	100
Showing Logic Values on Schematics	100
Using Query to Navigate	100
<u>Design Implementation</u>	101
Synthesizing and Compiling	101
Keeping Track of Processes	101
Understanding the Compilation Process	101
Compile (Logic or Schematic)	102
Compile Logic (for logic sources)	102
Compile Schematic (for schematic sources)	102
Compile EDIF (EDIF)	102
Check Syntax	102
Compiler Listing	103
Compiled Equations	103
Signal Cross-Reference (EDIF)	103
Process Options	103
Viewing and Setting Process Option Properties	103
Compiling Source Files	103
<u>Optimizing a Design</u>	104
Design Resources Check	104
Logic Synthesis Options	104
Boolean Logic Reduction	104
D/T Synthesis	104
Input Register Optimization	104
XOR Synthesis	104
Node Collapsing	105
Speed	105
Area	105
Fmax	105
Collapsing Max. Product Term	105
Collapsing Max. Input	105
Splitting Max. Product Term	105
Example	105

Example	106
Setting Logic Synthesis Options	106
Utilization Options	106
Logic Grouping	106
Fitting a Design	108
Assigning Pin and Node Locations	108
Pin and Node Pre-Assignment	108
Pin Assignment Guidelines	108
Large Functions at the End of a Block	108
Adjacent Macrocell Use	109
Modifying Assignments	109
Deleting Assignments	109
Ignoring Assignments	109
Power Control	109
Slew Rate Control	109
Partitioning	109
Balanced Partitioning	110
Fitting (Place and Route)	110
Placement	110
Spread Placement	110
Routing	110
Fitter Options	110
Pack Design	111
Spread Design	111
Advanced Options	111
Balance Partitioning	111
Spread Placement	111
Fitter Effort	111
Fitter Report Formats	111
Formatting the Fitter Report	112
The Fitter Report	112
Project Summary	112
Compilation Times	112
Design Summary	112
Device Resource Summary	112
GLB Resource Summary	113
GLB Control Summary	113
Optimizer and Fitter Options	113
Pinout Listing	113
(Input, Output, Bidir, Buried) Signal List	113
Signals Fan-out List	113
GLB (GLB name) Cluster Steering Tables	113
GLB (GLB name) Logic Array Fanin	113
Product Term Histogram	113
GLB Input Histogram	113
Post-Fit Equations	113
Backannotating Assignments	114
The Constraint Editor	115
Assigning Pins and Nodes	115
Assigning Signals to Groups	115
Node Preserving	116
Resource Reservations	116
Slew Rate	116
PULL	116
Open Drain	116

I/O Type	116
PLL	116
HSI	117
<u>The ispEXPLORER</u>	118
Overview of the ispEXPLORER	118
Use predefined or customized settings	118
Create multiple versions of design runs	118
Save the best constraints for your project	119
<u>The Report Viewer</u>	120
<u>Design Verification</u>	121
<u>The Waveform Viewer</u>	121
Opening the Waveform Viewer	121
Saving and Printing Waveforms	121
Printing Waveforms	121
Waveform Viewer Configuration	121
Waveform Display	122
Adding Waveforms	122
Finding the Signal You Want	122
Using the Probe Item Command	122
Duplicating Waveforms	123
Changing Waveform Locations	123
Hiding Waveforms	123
Creating Bus Displays	123
Expanding Bus Displays	123
Changing the Bus Radix	123
Moving Around	123
View Commands	123
Scroll Bars	124
Moving the Query Cursor	124
Marking Your Spot	124
Jumping to Events	124
Setting Signal and Bus Triggers	125
Analysis Techniques	125
Interaction with the Hierarchy Navigator	125
Displaying Simulation Values on a Schematic	126
Viewing Reports	126
<u>The Waveform Editor</u>	127
Bus Pulses	127
Patterns	127
Hierarchical Patterns	128
Editing Patterns	128
Simulation Time	128
Simulator Setup	128
Stimulus File Format	128
Saving Changes	128
<u>The Performance Analyst</u>	130
How does it Work?	130
Analysis Types	130
fMAX	131
Maximum Clock Operating Frequency	131
Default fMAX Path Trace	131
tSU	131
Setup Time	131

Default tSU Path Trace	131
Path Endpoints for tSU	132
Register D/T Inputs	132
Register CE Inputs	132
tPD	132
Propagation Delay Time	132
Default tPD Path Trace	132
tCO	132
Clocked Output-to-Pin Time	132
Default tCO Path Trace	132
tOE	133
Output Enable Path Delay	133
Default tOE Path Trace	133
tCOE	133
Clock to Output Enable Time	133
Default tCOE Path Tracing	133
Path Tracing Rules 133	
Tracing Enabled Through Bi-directional Paths	133
Tracing Enabled Through Register Asynch S/R Inputs	133
Tracing Enabled Through Transparent Latch D Inputs	134
Tracing Enabled Through Ripple Clocks	134
Batch Timing 134	
Running Timing Analysis in Batch Mode	134
Batch Commands	134
Set Operations	134
Path Tracing	135
Report	135
Switch Control	135
Batch File Example	135
Batch Command File Example	135
Device Programming	137
The ispVM System	137
Overview of ispVM System Software	137
Lattice Designs	137
Designs Compliant with IEEE 1532	138
Designs Compliant with IEEE 1149.1	138
Programming Basics	138
JTAG Scan Chains	139
Programming Algorithm Basics	139
Programming Times	140
USERCODE	140
I/O States During Programming	141
Programming Hardware	141
PC Hardware	142
ispDOWNLOAD Cable	142
ISP Engineering Kit Model 300	142
Programming Software	142
Programming on a Board Test System	143
Programming on JTAG Test Systems	143
Embedded Programming	143
The SVF Debugger	144
Understanding SVF Files	144
SVF Debugger Software Support of SVF Operations	144
The Model 300 Programmer	146

Overview	146
Device Support	146
Socket Support	146
Connector Support	146
Power Supply Support	147
Programming Software Support	147
Special Features	147
Calibration Control Switches	147
LOCAL	147
LSB, MSB, VCC ON	147
Software Control	147
Control Register	148

Overview of ispLEVER

The ispLEVER program offers an integrated environment consisting of several tools necessary to implement Lattice programmable devices. These tools are briefly described in the following paragraphs and covered in detail in other sections of this manual. They are listed in alphabetical order.

Constraint Editor

The Constraint Editor lets you specify pin and node location assignments, group assignments, I/O types settings, power level settings, resource reservations, PLL attributes, as well as output slew rates and JEDEC file options. The Constraint Editor reads the constraint file and displays the constraint settings. Modifications to the constraint file are made via the function dialogs. See the Constraint Editor for more information about this tool.

Hierarchy Browser

The Hierarchy Browser allows you to navigate through a design consisting of any combination of schematic and HDL modules. In contrast with the Hierarchy Navigator, the Hierarchy Browser works with designs whose top level is either a schematic or HDL source. Additionally, you can cross probe between design sources and their appropriate tool. See the Hierarchy Browser section for more information about this tool.

Hierarchy Navigator

The Hierarchy Navigator combines all the components of a multi-level design for viewing and analysis. You can traverse the full design, viewing each component in its full hierarchical context. See the Hierarchy Navigator section for more information about this tool.

ispEXPLORER

The ispEXPLORER lets you run multiple passes of your design using different combinations of Fitter/Optimizer settings and critical timing constraint to achieve the best solution. Results are summarized in a single spreadsheet and detailed reports for each run are accessible. See the ispEXPLORER section for more information about this tool.

ispVM System

The ispVM™ System software (ispVM) supports both serial and concurrent (turbo) programming of all Lattice devices in a PC environment. The ispVM System software is built around a graphical user interface. Device chains can be scanned automatically. Any required JEDEC files are selected by browsing with a built-in file manager. Non-Lattice devices that are compliant with IEEE 1149.1 can be bypassed once their instruction register length is defined in the chain description. Programmable devices from other vendors can be programmed through the vendor-supplied SVF file. See the ispVM System section for more information about this tool.

Lattice Logic Simulator

Lattice Logic Simulator performs logic simulation on your design before you implement it into a Lattice ispMACH or ispGDX device. You can observe not only the gate-level behavior at its inputs and outputs, but also the behavior of internal nodes. See Lattice Logic Simulator section for more information about this tool.

Library Manager

The Library Manager manages libraries of symbols that are used in Schematic Editor. The Library Manager lets you browse through the libraries and lets you maintain the libraries by adding, deleting, copying, or renaming the symbol files in the libraries. See the Library Manager section for more information about this tool.

Model 300 Programmer

The ISP Engineering Kit Model 300 programmer is an engineering device programmer that supports prototype development by allowing single-device programming directly from a PC. The Model 300 programmer supports all JTAG devices produced by Lattice, with device Vcc of 1.8, 2.5, 3.3, and 5.0V. See the Model 300 Programmer section for more information about this tool.

Performance Analyst

The Performance Analyst analyzes the performance of your design after it has been optimized and implemented by the Fitter. See the Performance Analyst section for more information about this tool.

Project Navigator

The Project Navigator is the primary interface for ispLEVER and provides an integrated environment for managing the project elements and processes, as well as accessing all ispLEVER tools. See the Project Navigator section for more information about this tool.

Report Viewer

You can use the Report Viewer to view, but not edit, the various report files generated by ispLEVER. See the Report Viewer section for more information about this tool.

Schematic Editor

The Schematic Editor is the ispLEVER schematic entry tool. It lets you create schematic (.sch) files that can represent a complete design or any component of a hierarchical design. See the Schematic Editor section for more information about this tool.

SVF Debugger

The SVF Debugger can be used with ispVM System software to help you debug a Serial Vector Format (SVF) file. The SVF Debugger software allows you to program a device, and then edit, check syntax, debug and trace the

process of an SVF file. See the SVF Debugger section for more information about this tool.

Symbol Editor

The ispLEVER software comes with a standard symbol library. Use the Symbol Editor to create symbols or primitive elements that represent an independent schematic module. You can also use the Symbol Editor to create decorative symbols, such as title blocks. See the Symbol Editor section for more information about this tool.

Text Editor

The Text Editor is the ispLEVER text entry tool. You use this tool to create and edit text-based files, such as ABEL-HDL files, test stimulus files, and project documentation files. See the Text Editor section for more information about this tool.

Waveform Editor

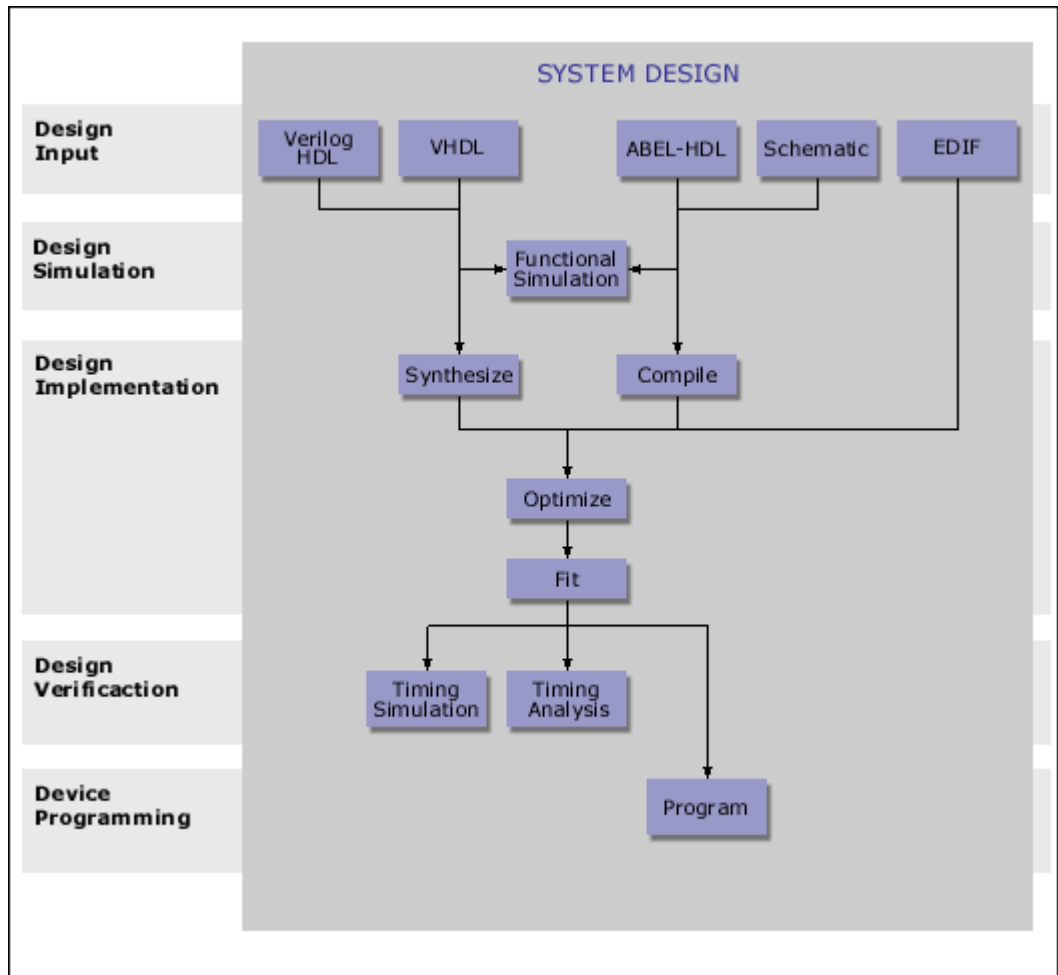
The Waveform Editor lets you create the stimulus graphically, by clicking and dragging with the mouse. You see exactly what each waveform will look like, as well as its timing relationship to all the other waveforms. See the Waveform Editor section for more information about this tool.

Waveform Viewer

The Waveform Viewer lets you view the results of simulation. You can display the waveform of any net in your design. The Waveform Viewer is fully interactive with the Hierarchy Navigator; clicking on a net in the schematic automatically displays its waveform. See the Waveform Viewer section for more information about this tool.

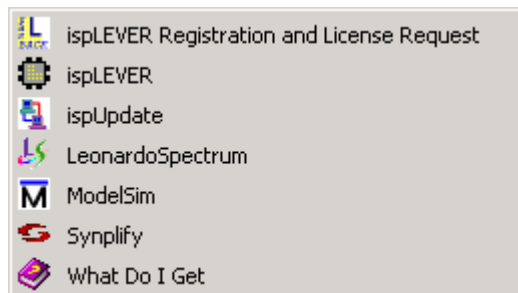
CPLD Process Flow

The ispLEVER CPLD process flow diagram.



Program Folder

When you install the ispLEVER software, several icons are created in the Lattice Semiconductor Programs folder. The contents of the folder may vary from the picture below, depending on your installation choices.



Learning ispLEVER

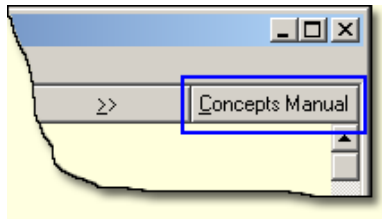
Lattice ispLEVER includes a variety of resources to help you learn the software quickly and become proficient in designing, implementing, verifying, and programming your design in a Lattice device. These resources include a Concepts manual, comprehensive online Help, an online HTML-based tutorial with interactive lessons covering a variety of practical areas, and many example design projects to practice developing your skills.

ispLEVER Concepts

The ispLEVER Concepts is a resource manual of design, implementation, verification, and programming concepts that comprise ispLEVER v1.0. This manual explains the purpose of each ispLEVER tool, and gives guidelines for using these tools to implement Lattice programmable devices.

The contents of this document are also covered in ispLEVER Help. However, this document offers the benefit of being able to print multiple pages (topics) at a time.

You can access this manual from the **Project Navigator: Help > Manuals** command, or from the toolbar of ispLEVER Help.



ispLEVER Help

The ispLEVER Help is a comprehensive, hyper-linked information system covering all aspects of the ispLEVER software. Following the Introduction, ispLEVER Help is divided into seven parts, each representing a major step in the Lattice programmable device design flow. Within each part are tool and process sections. Each of these sections is organized with the concepts, procedures, and references you will need to successfully complete your design. Additional reference information is provided covering a wide range of topics.

ispLEVER Tutorial

The ispLEVER Tutorial contains three tutorials designed to benefit all ispLEVER software users and is the best place to start if you want to get some hands-on experience. By working through the tutorial lessons, you'll learn how to create sample design projects with some of ispLEVER's most useful and powerful features.

Example Projects

The ispLEVER software comes with a broad range of example projects. These examples include a combination of Schematic/ABEL, Schematic/Verilog-HDL, Schematic/VHDL, and EDIF project types. You can access the examples from the **Project Navigator: File > Open Example** command.

Third-Party Documentation

The ispLEVER software tightly integrates several third-party synthesis and simulation tools. In addition to design flow documentation included in ispLEVER Help, you can find a complete manual set for these products supplied by their respective vendor from the **Project Navigator: Help > Manuals** command.

Product Literature

Lattice maintains a comprehensive selection of online product literature in Adobe PDF format. This includes documents such as product brochures, data sheets, design techniques, application notes, and packaging information. You can download these documents from the Lattice Web site (<http://www.latticesemi.com>).

Using ispUPDATE

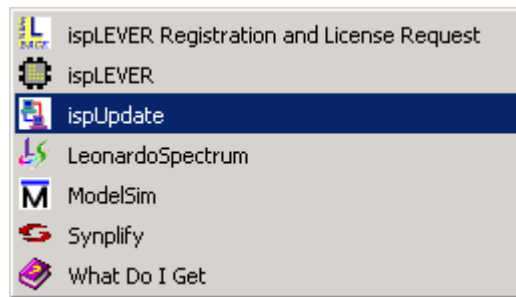
The ispUPDATE program lets you select the version of ispDesignEXPERT or ispLEVER to be updated when there are multiple versions of the software installed on the same computer. The ispUPDATE software is a stand-alone application; you do not need to run the Project Navigator to implement the update.

Starting the ispUpdate Program

After you start the ispUPDATE program, you can set your Internet communication options.

To start *ispUPDATE*:

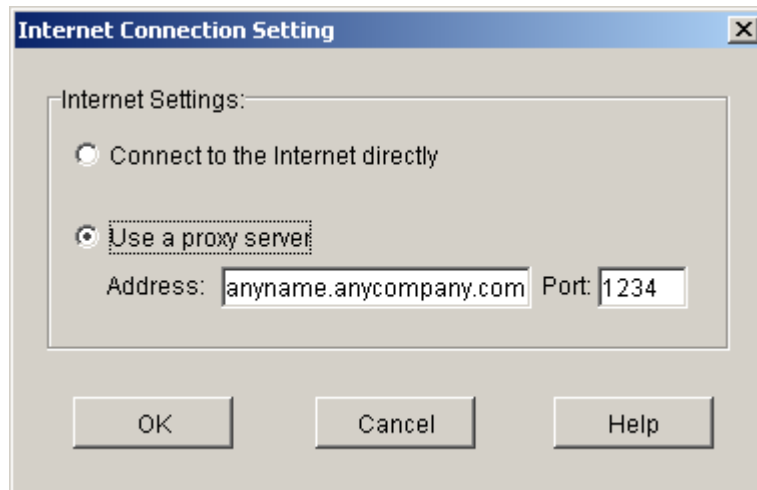
1. From the Lattice program menu, select **ispUPDATE** to open the program.



2. In the Select Software list, select the Lattice software program that you want to update.

Note: If the selected software is not installed on your computer, the Update button is disabled. See the Download section later in this manual.

3. Click **Internet Setting** to open the Internet Connection Setting dialog. Choose one of the following options, and then click **OK**.



If your organization has a firewall, your browser may need to go through a proxy server before connecting you to the Internet. The proxy server prevents outsiders from breaking into your organization's private network.

- **Connect to the Internet directly** – Select this option if you do not have to go through a proxy server.
- **Use a proxy server** – Select this option if you have to go through a proxy server. Ask your system administrator for the URL address and port assignment.

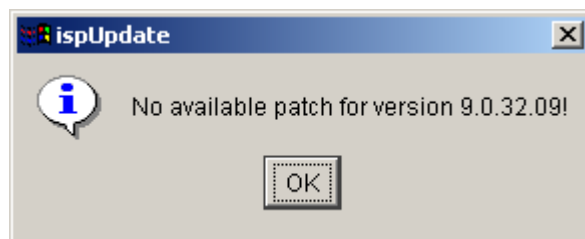
Using the Update Feature

Use the Update feature when you want to update your currently installed Lattice software.

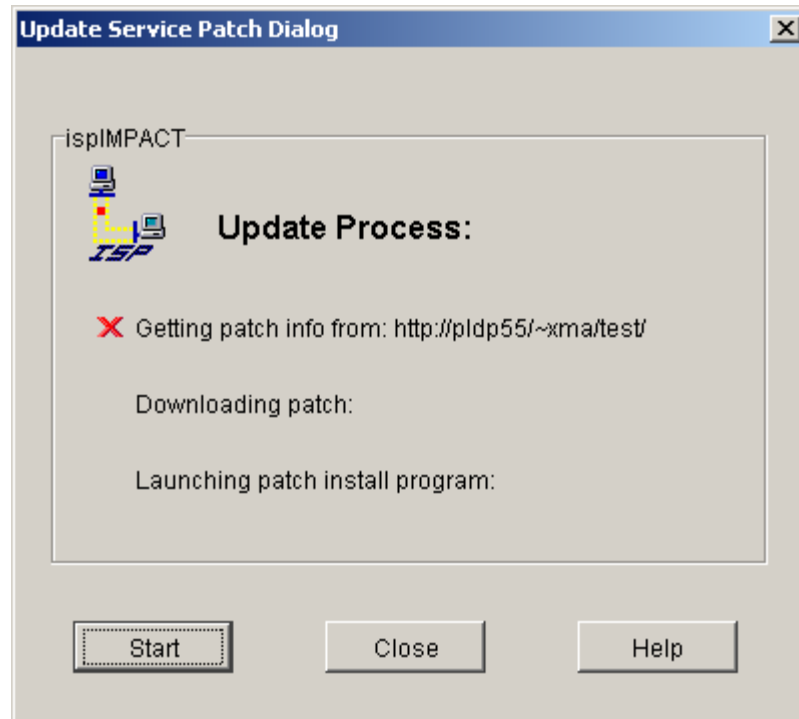
To use Update:

1. Click **Update**.

The ispUPDATE software automatically determines the installed version of the selected Lattice software program. If there are no service patches available for the chosen release, a message box appears saying so.

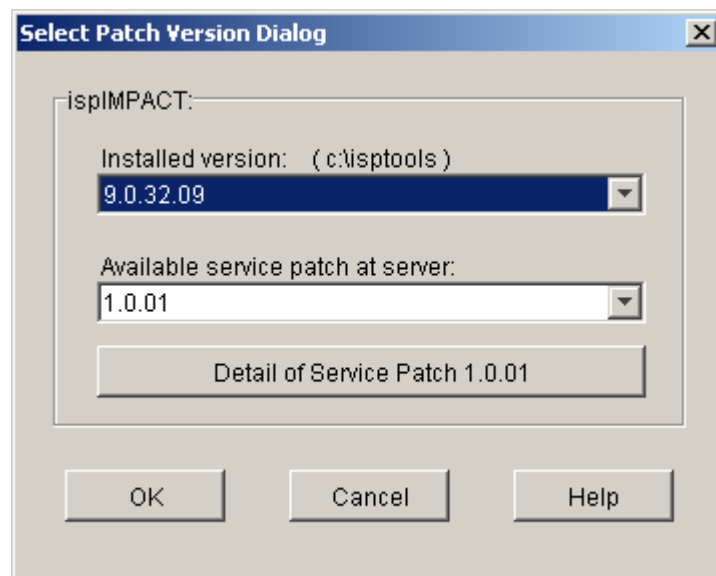


2. Click **OK** to close the ispUpdate dialog box.



3. Because no Service Patches are available for this installed version, click **Close** to close the dialog box.

If a Service Patch is available for the installed version, the Select Patch Version dialog box opens.

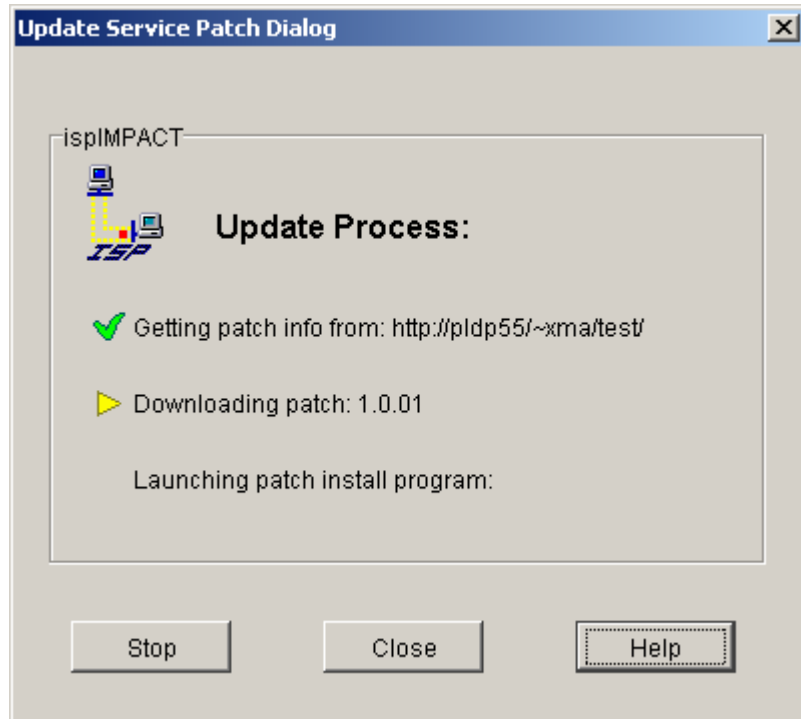
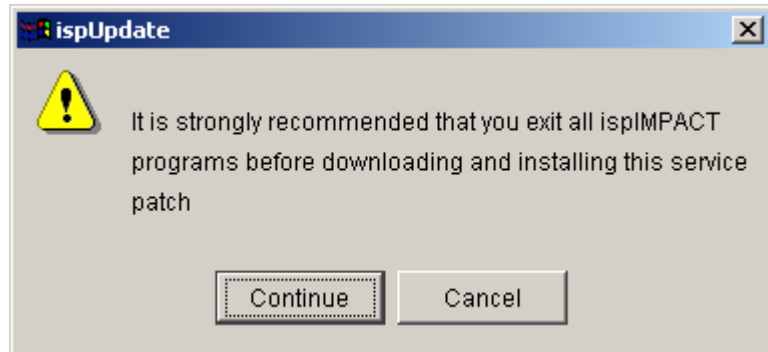


4. In the dialog box, do the following:
 - Select the Installed version you want.
 - Select the service patch you want.

- Optionally, click **Details of Service Patch** to open your default browser and view the description of the service patch.

Note: If no services patches are available for the selected installed version, you will not see a button under the list boxes labeled Detail of Service Patch.

- Click **OK** to close the dialog box.
5. The message dialog appears. Click **Continue**. The ispUPDATE program begins downloading the service patch and automatically launches the installation program.

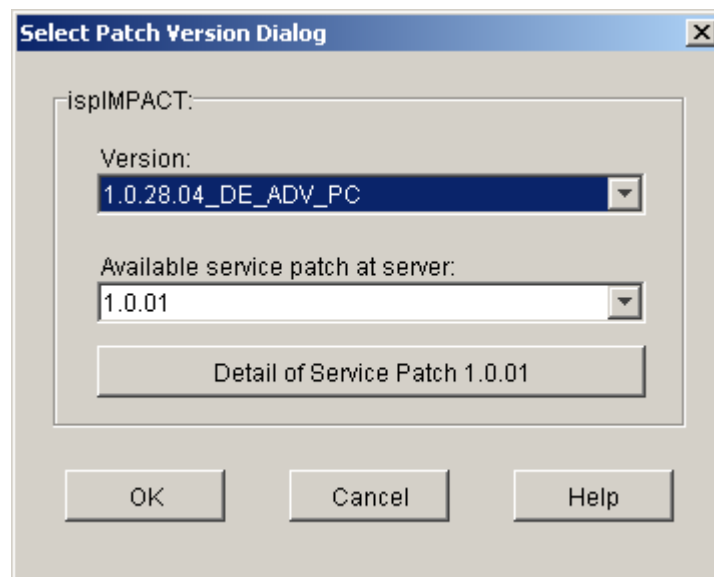


Using the Download Feature

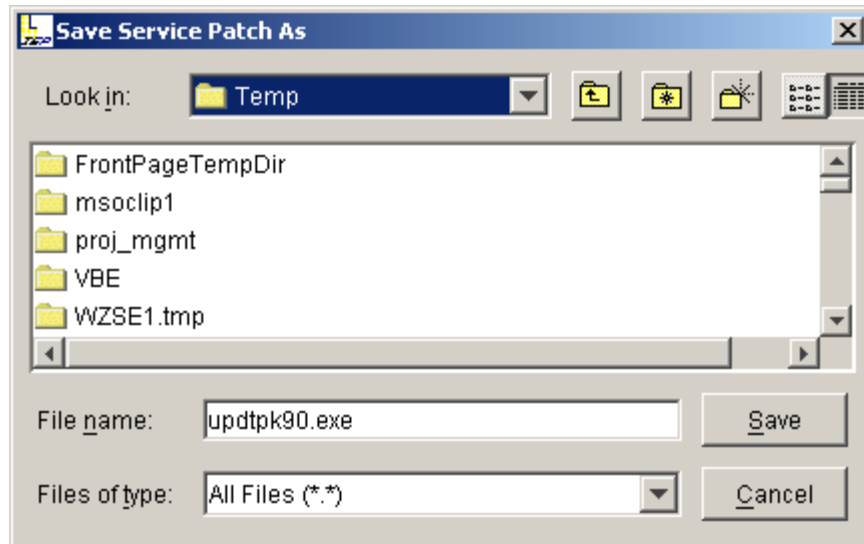
Use the Download feature when you want to download any of the total available service patches for all Lattice software releases, regardless of whether they are installed on your computer.

To use download:

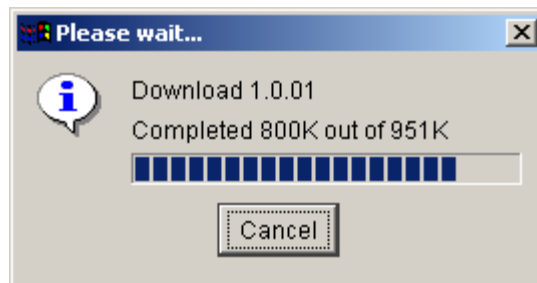
1. Click **Download** to open the Download Service Patch dialog and the Select Patch Version dialog box.
2. In the Select Patch Version dialog box, do the following:



- Select the software version you want.
- Select the service patch you want
- Optionally, click Details of Service Patch to open your default browser and view the description of the service patch.
- Click OK to open the Save Service Patch dialog box.



3. Navigate to a location on your computer where you want to save the file, and then click **Save**. The ispUPDATE software starts downloading the service patch.



Launching the Patch Install Program

After you have downloaded the service patch, you can install it to update your Lattice software.

To install the service patch:

1. Go to the location where you saved the service patch.
2. Double-click the service patch file (`filename.exe`) and follow the on-screen setup instructions.

Migrating from ispDesignEXPERT 8.x (8.2, 8.3, or 8.4) to ispLEVER

The new ispLEVER software gives you an improved and more unified constraint flow. You must convert legacy ispDesignEXPERT 8.2, 8.3, or 8.4 constraint files to ispLEVER constraint files before running your design in ispLEVER. To convert the constraint files, you can use the **Legacy Constraints Translator** on the Lattice Semiconductor program menu, or the **legacy2lci** command line utility provided with the ispLEVER software, located in `<install_path>\ispcpld\bin`.

MACH Devices

For MACH4 and MACH5 devices, designs created with ispDesignEXPERT 8.2, 8.3, or 8.4 use a `.vci` constraint file, which must be converted to an `.lci` constraint file for ispLEVER.

ispLSI Devices

For ispLSI 1K and 2K devices, designs created with ispDesignEXPERT 8.2, 8.3, or 8.4 use a parameter file (`.par`), a property file (`.prp`), a pin file (`.ppn`), a user code file (`.ues`), a pin reservation file (`.rsp`), or a fitting options setting file (`.stp`), all of which can be converted to a single `.lci` file for ispLEVER. The new ispLEVER software gives you an improved and more unified constraint flow. You must convert legacy ispDesignEXPERT 8.2, 8.3, or 8.4 constraint files to ispLEVER constraint files before running your design in ispLEVER. To convert the constraint files, you can use the **Legacy Constraints Translator** on the Lattice Semiconductor program menu, or the **legacy2lci** command line utility provided with the ispLEVER software, located in `<install_path>\ispcpld\bin`.

Note: Before beginning the conversion process, be sure to save an archive copy of your entire project directory.

Converting Files Using the Legacy Constraints Translator

The Legacy Constraints Translator is a graphical user interface application that reads the `<projectName>.syn` file, and other files according to the optional parameters.

To convert files using the Legacy Constraints Translator:

1. Archive your project directory so that you will have a backup.
2. From the Lattice program menu, select **Legacy Constraints Translator** to open the program.
3. In the dialog, do the following:
 - In Project Name, select the ispDesignEXPERT 8.x project that you want to convert.

- Under Select Device, the default is the current target device. To select a different device, you must change the project name in the Project Name field. If the target device is not supported, all options are disabled.
- Under Options, select the files that you want to convert.

Note: For ispLSI devices, all options are available. For MACH devices, no options are available.

4. Click **OK**. The selected legacy files are converted to ispLEVER files in the specified project directory.

Converting Files Using the legacy2lci Command Line Application

You can use the legacy2lci command line utility to specify constraint file conversion options.

To convert files using the legacy2lci command line utility:

1. Archive your project directory so that you will have a backup.
2. For MACH4 or MACH5 devices, using the Windows Command Prompt, change directory (cd) to your project directory, and type (on one line):

```
<install_path>\ispcpld\bin\legacy2lci
<project_name>
```

For ispLSI 1K and 2K devices, using the Windows Command Prompt, change directory (cd) to your project directory, and then type (on one line):

```
<install_path>\ispcpld\bin\legacy2lci
<project_name> [-prp <filename_ext>] [-p [-par
<filename_ext>]] [-n [-ppn <filename_ext>]] [-u
[-ues <filename_ext>]] [-r [-rsp
<filename_ext>]] [-x [-xct <filename_ext>]] [-
ppnOnly <filename_ext>]
```

where [] denotes optional parameters for 1K and 2K devices:

-prp converts a user property file name that differs from the project name, <project_name>.prp. It is recommended that you not use the 8.4 system-created _edif.prp file. However, you can use the _edif.prp file in those cases where the <project_name>.prp file is 0 bytes or empty.

-p converts the PAR file. By default, the PAR file is <project_name>.par. When you use the -p option, the Part Name, PRP, and PPN defined in the PAR file are ignored.

-par is used to specify a PAR file name that differs from the project name, <project_name>.par.

-n converts the PPN file. By default the PPN file is <project_name>.ppn. When you use the -n option, the pin

assignments in the PPN file are used, and those defined in the PRP file are ignored.

-ppn is used to specify a PPN file name that differs from the project name, `<project_name>.ppn`.

-u converts a UES file.

-ues is used to specify a UES file name that differs from the project name, `<project_name>.ues`.

-r converts an RSP file.

-rsp is used to specify an RSP file name that differs from the project name, `<project_name>.rsp`.

-x converts an XCT file.

-xct is used to specify an XCT file name that differs from the project name, `<project_name>.xct`.

-ppnOnly is used to override all of the above parameters. When using ppnOnly, legacy2lci does not need a PRP and SYN file. The part name is derived from the PPN itself.

Note: For ispLSI 1K and 2K devices, the part name defined in `<project_name>.syn` will be used; therefore, the part name defined in PAR file will be ignored.

3. After the conversion process, make sure that the converted `.lci` file is in your working project directory.
4. In ispLEVER, choose **File > Open** to open your Project (`.syn`) file.
5. Save the project by choosing **File > Save**.

Notes:

- After you perform design migration using the legacy2lci utility, you must re-synthesize your design with your ispLEVER synthesis tool and import the newly synthesized EDIF file into ispLEVER. You cannot import legacy EDIF files that were synthesized in ispDesignEXPERT 8.x.

- For legacy schematic designs, you may need to map old library symbols to equivalent new library symbols in ispLEVER after using the legacy2lci utility. Contact Lattice Technical Support for help with mapping old library schematic symbols with new library schematic symbols.

Using ispLEVER Help

Following the Introduction, ispLEVER Help is divided into seven parts, each representing a major step in the Lattice programmable device design flow. Within each part are tool and process sections. Each of these sections is organized with the concepts, procedures, and references you will need to successfully complete your design. Additional reference information is provided covering a wide range of topics.

The ispLEVER Help uses the WinHelp 2000 tri-pane window, which includes:

- **Button bar.** The button bar is displayed across the top of the ispLEVER Help window. You can show or hide the Navigation pane by toggling the Help Topics button. Other buttons take you Back to the previous topic, let you Print the current topic, and browse backward (<<) or forward (>>) in topic sequence.
- **Navigation pane.** The navigation pane takes up approximately one-third of the left side of the ispLEVER Help window (or one-third of the total window size). It contains three tabs: the Contents tab, the Index tab, and the Search tab. The Contents tab uses the books and pages metaphor to display contents and remains synchronized with the topic displayed in the Topic pane. The Index tab provides access to the Help index, and the Search tab provides access to full-text search. You do not have to leave the Contents, Index, or Search tabs to see their selected topic; it appears in the Topic pane on the right.
- **Topic pane.** Help topics appear on the right side of the ispLEVER Help window. If the topic extends beyond the window, scroll bars allow you to see the rest of the topic.

Context-sensitive Help

You can use context-sensitive Help in two ways: clicking **Help** buttons and pressing the **F1** key on your computer's keyboard. Most ispLEVER software dialogs have a Help button. Clicking this button or pressing the F1 key opens a relevant help topic. For help on menu commands, place your cursor over the command name and press the F1 key.

Running ispLEVER from the Command Line

You can run the ispLEVER software from the command line on PC and UNIX using ispflow command line software. The ispflow software will attempt to fit the design to the specified part.

Contacting Lattice Semiconductor

Please use the following address and numbers to contact Lattice Semiconductor.

Headquarters

Lattice Semiconductor Corporation
5555 NE Moore Ct.
Hillsboro, OR 97124

Phone: (503) 268-8000
Fax: (503) 268-8037

Product Support

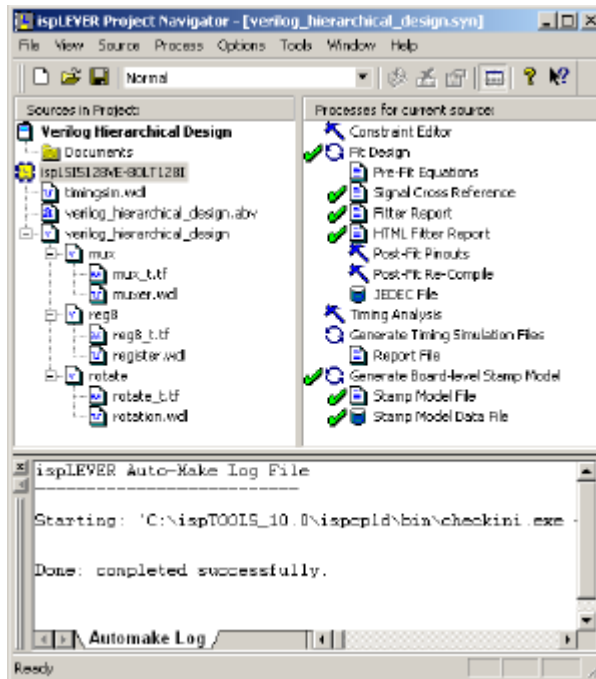
Lattice is dedicated to customer satisfaction. If you have questions about ispLEVER, please contact your sales representative or visit Technical Support on our Website.

Note: *If your browser is not responding to this link, please copy the URL:
<http://www.latticesemi.com/support/index.cfm>.*

Project Management

Project Navigator

The Project Navigator is the primary interface for the ispLEVER software. Using the Project Navigator, you can select all the source components for a design, as well as specification documents and test files, and assemble them into one project file.



The Project Navigator Interface

The Project Navigator has two primary interface elements: the Sources window and the Processes window.














Using the Project Navigator, you can select all the source components for a design, as well as specification documents and test files, and assemble them into one project file. The Project Navigator also helps you keep track of all the processing steps necessary to move the design from conception through implementation of a programmable device. When you switch the target device, the Project Navigator automatically changes the design flow and processes to appropriate ones for the new target device.

The Project Navigator also associates all the tools needed for a particular design step. For example, for HDL source files, the Project Navigator associates the Text Editor and HDL synthesis tools; for schematic sources, the Project Navigator associates the Schematic Editor, Symbol Editor, Hierarchy

Navigator, Library Manager tools, and schematic compiling tools; and for waveform stimulus source files, the Project Navigator associates Waveform Editor, Waveform Viewer, and Lattice Logic Simulator tools. Furthermore, the Project Navigator keeps track of your preferences, automatically setting options that work for most systems until you want to modify the options yourself.

Valid Source Types

Below are the acceptable sources for a project. A new project will not have any sources except for the Project Notebook.

Source Type	Icon	File Extension
Project Notebook		None
Target Device		None
Document Source		.wri, .doc, .txt, .xls, .hlp (or any extension not recognized by the Project Navigator)
Schematic Source		.sch
ABEL-HDL		.abl
ABEL-HDL Test Vector		.abv
VHDL		.vhd
Verilog HDL		.v
EDIF Netlist		.ed*
Waveform Stimulus		.wdl
VHDL Test Bench		.vhd
Verilog HDL Test Fixture		.tf
Undefined or incorrect		Any source reference

Source Hierarchy

One of the source files in a project is the top-level source for the design, which can be an HDL module or schematic. The top-level source defines the inputs and outputs that will be mapped into the device, and references the logic descriptions contained in lower-level sources. Referencing another source is called instantiation.

Lower-level sources can also instantiate sources 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.

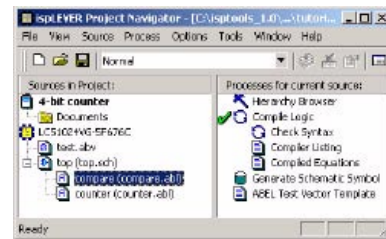
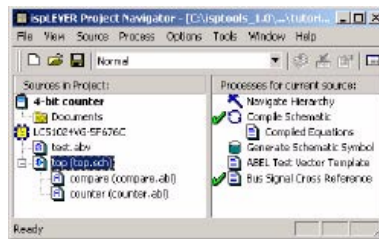
Process Flows

One of the most powerful features of ispLEVER is the context-sensitive nature of the Project Navigator. It automatically adjusts the processes for you depending on what you do.

The steps in the Processes window are context-sensitive in two ways. First, the process flow changes depending on the type of source file selected in the Sources window (source-level flow). Second, the processing for a given file changes depending on the target device (project-level flow).

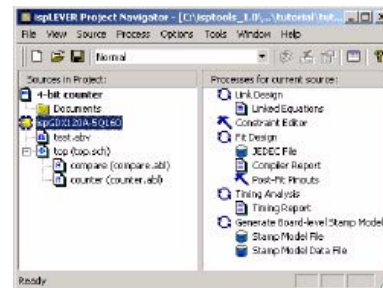
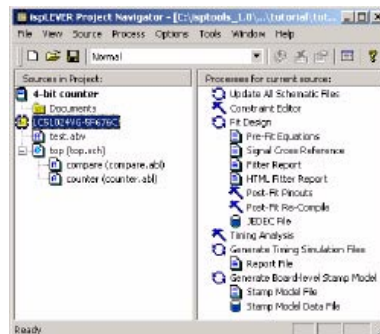
Source-Level Flow

Notice that the processes are different in these two examples depending on the source that is selected. The example on the left shows the processes for the selected schematic source, while the example on the right shows the processes for the selected ABEL source.



Project-Level Flow

This example shows the difference in processes depending on the target device. In the example on the left, the target device is an ispMACH 5000VG. The example on the right uses an ispGDX device.



Initialization

At the beginning of a new project, the ispLEVER software automatically copies a default constraint file from the ispLEVER directory into your project directory. For first-time users, the default settings allow most designs to

achieve a First-Time Fit (FTF). For users requiring more control, the default settings can be easily changed to achieve better fitting density or performance.

You can control the contents of the constraint file using the Global Constraints dialog box and the Location Assignments dialog box.

Using the Global Constraints Dialog Box to Control Optimization

Using the Global Constraints dialog box, you can pack your design, spread your design, or use other advanced options such as specifying device utilization levels.

Using the Location Assignments Dialog Box to Pre-assign Pins and Nodes

The Location Assignments dialog box lets you specify pin and block locations, group signals in specific blocks, or even reserve pins for later use.

Describing a Project

You "describe" a project by targeting a particular device implementation and by specifying the project files that will represent the design.

Design Hierarchy

A single module design is a flat design containing only one source describing the entire design such as a schematic. You can also have a test file such as ABEL-HDL test vectors in a flat design because all processes such as functional simulation in the flat design involve the entire design.

When designs are in multiple levels, this is called hierarchical design. The ispLEVER software supports full hierarchical design to clarify its function or permit the reuse of functional blocks.

Tips for Defining Projects

Use the following guidelines when saving and naming source files and your project:

- Understand and use the different methods described in the hierarchical design section.
- Avoid using ABEL, VHDL, Verilog, or EDIF reserved words for module and signal names in any source files.
- You can instantiate the source to use the same source many times in a design, but two different sources with the same name can cause problems with the hierarchy. For example, do not use an ABEL-HDL source called *compare* and a schematic source also called *compare*.

Where Source Files are Placed

After you import a source into the Project Navigator, it appears in the Sources window. However, where the source appears in the window depends on the following:

- If the imported source is a documentation file or a file type not recognized as a logic description or test file, the source appears between the Notebook icon and Target Device icon.
- If the source is a logic description, the source is placed in alphabetical order for each level of hierarchy following the project notebook and the targeted device. For example, if the source is called `multiplx` and the top-level source, a schematic called `myChip`, contains a functional block called `multiplx`, the source is placed underneath `myChip` in the Sources window.
- If the source is an ABEL test vector file, the source is placed beneath the Target Device icon.

Processing a Design

A process is a specific task in the overall processing of a source or project. Typical processing tasks include netlisting, compiling, logic reduction, logic synthesis, fitting the design, and simulation. To view the available processes for a source, select the source. Then the ispLEVER software displays the processes for that source in the Processes window.

Tip for Saving and Naming Projects

Do not save more than one project in the same directory when saving and naming source files and projects.

Forcing a Process to Run

If the process is updated (indicated by a check mark to the left of the process), it will not run again. However, you can force a process to run by doing the following.

- Choose Process > Force to start the highlighted process and run all the intermediate steps, even if the process is up-to-date. When the process has finished running, the Project Navigator displays the selected file, if applicable. This command allows you to temporarily override the Process Force settings in the Environment Options dialog box to start the highlighted process.
- Choose Process > Force One Level to start the highlighted process.

Reserved File Names

The ispLEVER software reserves several file name extensions for its own use. You should avoid using the following extensions when naming your own files:

_ln	Hierarchy Navigator log file
_sc	Schematic Editor log file
_sy	Symbol Editor log file
_wt	Waveform Editing Viewer log file
_wv	Waveform Viewer log file
.asc	ASCII schematic file
.asy	ASCII symbol file
.bin	Binary waveform file
.ed*	EDIF netlist
.err	Error OUTPUT file
.his	Waveform Viewer history file
.nam	Binary waveform name file
.pin	Netlist file for generic netlist by pin
.sch	Schematic Editor files
.sym	Symbol Editor file
.tre	Hierarchy Navigator file
.lci	Constraint file
.lct	Temporary working copy of the constraint file
.lco	Constraint output from the Fitter (such as the post-fit pinouts, etc.)
.vtr	Hierarchy Navigator temporary file
.wav	Waveform Viewer waveforms and trigger information
.wdl	Waveform Editing Tool database
.wet	Waveform Editing Tool database

Pop-up Menus

Pop-up (right-click) menu support helps accomplish frequently used tasks. Most of the commands on the pop-up menus are also available from the toolbar or menus.

Hierarchical Design

What is a Hierarchical Design?

A design with more than one level is called hierarchical. A single-level design is referred to as being flat. Converting a section of circuitry to a block makes a flat design hierarchical. This is commonly referred to as "hierarchical" design.

The ispLEVER software supports full hierarchical design. Hierarchical structuring permits a design to be broken into multiple levels, either to clarify its function or to permit the reuse of functional blocks. For instance, a large, complex design does not have to be created as a single module. By using a hierarchical design, each component or piece of a complex design can be created as a separate module.

A design is hierarchical when it is broken up into functional blocks, or modules. For example, you could create a top-level schematic describing an integrated circuit. In the schematic, you could place a Block symbol (a Block symbol represents a functional block) that provides a specific function of the chip. You can then elaborate the underlying logic for the Block symbol as a separate schematic or as a separate HDL module.

The module represented by the Block symbol is said to be at one level below the schematic in which the symbol appears. Or, the schematic is at one level above the Block's module. Regardless of how you refer to the levels, any design with more than one level is called a hierarchical design.

Advantages of Hierarchical Design

The most obvious advantage of hierarchical design is that it encourages modularity. A careful choice of the circuitry that composes your module will give you a Block symbol that can be reused.

Another advantage of hierarchical design is the way it lets you organize your design into useful levels of abstraction and detail. For example, you can begin a project by drawing a top-level schematic that consists of nothing but Block symbols and their interconnections. This schematic shows how the project is organized but does not display the details of the modules (Block symbols).

You then draw the schematic for each Block symbol. These schematics can also contain Block symbols for which you have not yet drawn schematics. This process of decomposition can be repeated as often as required until all components of the design have been fully described as schematics.

Breaking the schematic into modules adds a level of abstraction that lets you focus on the functions (and their interaction) rather than on the device that implements them. At the same time, you are free to view or modify an individual module.

Although there are many ways of "breaking apart" a complex design, some may be better than others. In general:

- Each module should have a clearly defined purpose or function and a well-defined interface.

- Look for functions or component groupings that can be reused in other projects.
- The way in which a design is divided into modules should clarify the structure of the project, not obscure it.

Hierarchy vs. Sheets

Creating a hierarchical design is not the same as creating a schematic with multiple sheets. In a schematic, you can add as many sheets as desired to extend beyond the original sheet. However, regardless of how many sheets you add, all the components of the design are still at a single level; all sheets are still contained in the same module.

Approaches to Hierarchical Design

Hierarchical designs consist of one top-level module. This module can be of any format, such as ABEL-HDL, VHDL, Verilog HDL, schematic, or EDIF netlist. Lower-level modules can be of any supported sources and are represented in the top-level module by functional blocks or other "placeholders."

Following are some rules you need to follow when creating a hierarchical design in ispLEVER.

- The top-level source can be of any format, such as ABEL-HDL, VHDL, Verilog HDL, schematic, or EDIF netlist.
- For hierarchical Schematic/ABEL designs:
 - If the upper-level source is a schematic file, the lower-level source can be either a schematic or an ABEL-HDL file.
 - If the upper-level source is an ABEL-HDL file, the lower-level source can be either a schematic or an ABEL-HDL file.
- For hierarchical Schematic/VHDL designs:
 - If the upper-level source is a schematic file, the lower-level source can be either a VHDL file or a schematic file.
 - If the upper-level source is a VHDL file, the lower-level source can only be a VHDL file.
- For hierarchical Schematic/Verilog HDL designs:
 - If the upper-level source is a schematic file, the lower-level source can be either a Verilog HDL file or a schematic file.
 - If the upper-level source is a Verilog HDL file, the lower-level source can only be a Verilog HDL file.
- For EDIF designs:
 - Hierarchical EDIF design is not allowed.

You can create the top-level module first, or create it after creating the lower-level modules. For example, in the Schematic Editor you can create schematic project components in any order and then combine them into a complete design. You can draw a schematic first and create a Block symbol for it

afterwards; or you can specify the Block first and create the schematic for it later.

Hierarchical ABEL-HDL Design

You can use ispLEVER to specify a lower-level block symbol in an ABEL-HDL design. Also, you can instantiate a lower-level schematic Block symbol in an ABEL module.

Hierarchical Schematic Design

You can use the ispLEVER software to specify a lower-level block symbol in a schematic, or you can instantiate a lower-level ABEL-HDL block symbol in a schematic.

Hierarchical Verilog HDL Design

You can use the ispLEVER software to specify a lower-level schematic block symbol in a Verilog HDL module, or you can instantiate a lower-level Verilog HDL block symbol in an upper-level Verilog HDL.

Hierarchical VHDL Design

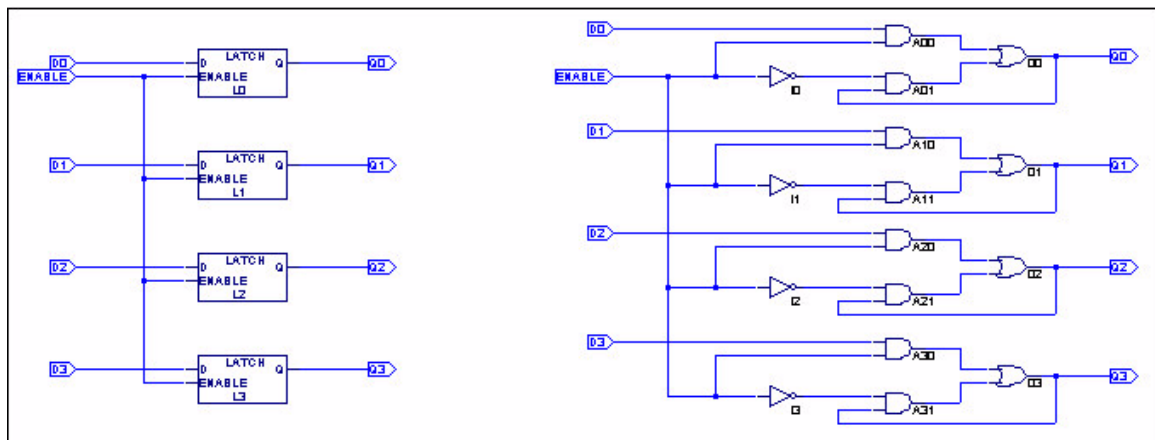
You can use the ispLEVER software to specify a lower-level schematic block symbol in a VHDL module, or you can instantiate a lower-level VHDL block symbol in an upper-level VHDL.

Hierarchical Design Considerations

Apply the following considerations when using hierarchical design techniques. We take a hierarchical schematic design as an example (shown below). The rules implied in the example are also applicable to other types of hierarchical designs.

- Hierarchical Design Structure
- Hierarchical Naming
- Nets in the Hierarchy
- Automatic Aliasing of Nets

Example: REG4 and its Equivalent Circuit



Hierarchical Design Structure

When a symbol is placed in a schematic, the component or subcircuit that the symbol represents is added to the circuit. For example, when you place a latch symbol, you are actually including the OR gate, inverter, and two AND gates from the latch's schematic.

The example shows a 4-bit register (REG4) constructed from four latch symbols (latch.sym). The right side of the figure shows the underlying components. The four latch symbols represent a total of eight AND gates, four OR gates, and four inverters.

This hierarchical building process could be repeated by using the Schematic Editor's **File > Matching Symbol** command or **File > Generate Symbol** command (if the corresponding .naf file has been generated) to create a symbol for schematic reg4, and then placing the reg4 symbol in a higher-level schematic. If you created a schematic for a 16-bit register, reg16, by placing four copies of symbol reg4, you would be defining a circuit with a total of 64 gates. But instead of having to view 64 gates on a single level, you can work with symbols that represent gates, at the appropriate level of detail.

Hierarchical Naming

In the `latch` schematic example, the inverter has the instance name `I1`. In schematic `reg4`, four copies of the symbol `latch` are placed and assigned instance names `L1` through `L4`. Schematic `reg4`, therefore, contains four copies of inverter `I1`.

The Hierarchy Navigator distinguishes among these otherwise identical inverters by combining the inverter's instance name with the instance name of the latch containing it. The four inverters are therefore named (in the Hierarchy Navigator):

`L1.I1`

`L2.I1`

`L3.I1`

`L4.I1`

If we created a 16-bit register by combining four `reg4` symbols, the resulting schematic would represent a new hierarchical level containing four copies of `reg4` (named `R1` through `R4`). Each copy of `reg4` contains the four inverters as named above. The Hierarchy Navigator would then name the 16 inverters by combining the instance names of the four `reg4` symbols with each of the four instance names of the inverters as follows:

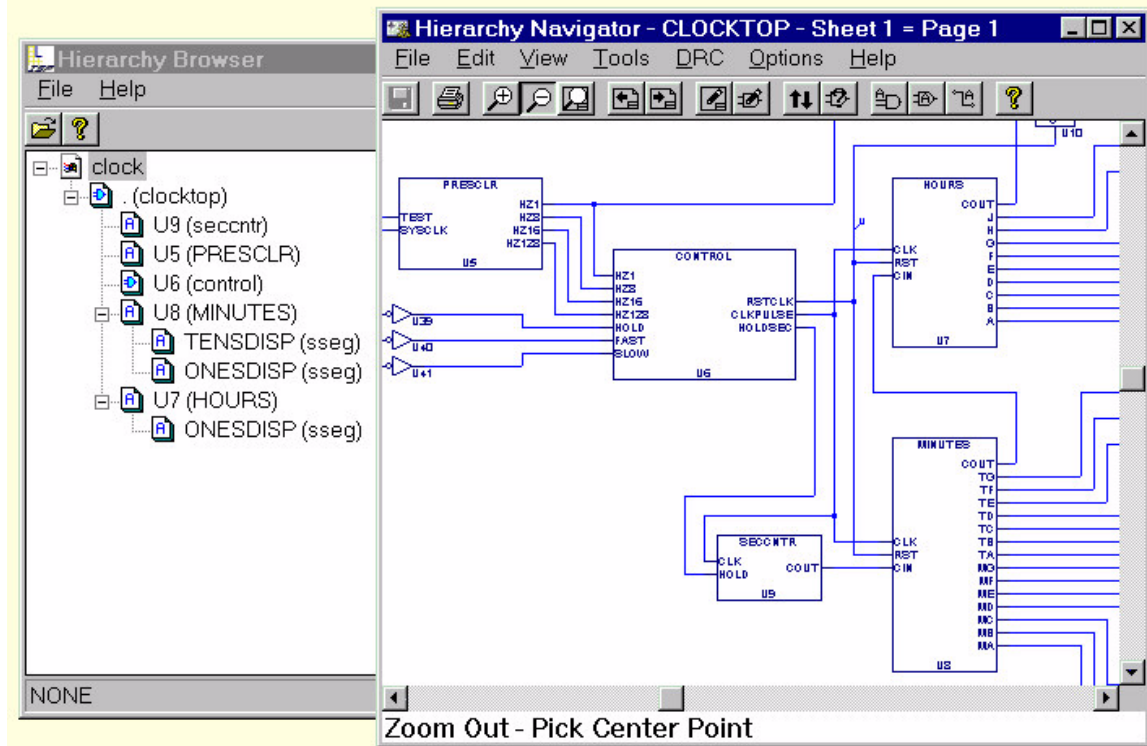
`R1.L1.I1`, `R1.L2.I1`, `R1.L3.I1`, `R1.L4.I1`

`R2.L1.I1`, `R2.L2.I1`, `R2.L3.I1`, `R2.L4.I1`

`R3.L1.I1`, `R3.L2.I1`, `R3.L3.I1`, `R3.L4.I1`

`R4.L1.I1`, `R4.L2.I1`, `R4.L3.I1`, `R4.L4.I1`

When you view an individual latch schematic in the Schematic Editor, you see the instance names of the gates, without the hierarchical context. When the schematic becomes part of a larger design and is viewed in the Hierarchy Navigator, the instance names include the hierarchical path (as shown above) to assure their uniqueness.



Nets in the Hierarchy

The schematic definition for the latch circuit contains both local and external nets. The output of the inverter is connected to the AND gate with a local net. Two other local nets connect the outputs of the AND gates to the inputs of the OR gate. Assume these nets have been named N1, N2, and N3. When 16 copies of this circuit are combined in `reg16`, 16 copies of these local nets are created.

The 16 local nets named N1 are individual nets, not branches of the same net, so the Hierarchy Navigator creates a unique name for each. The local net name (N1) is prefixed with the instance name of the schematic where the net is defined. A dash separates the net and instance names. The 16 N1s then become:

R1.L1-N1, R1.L2-N1... R4.L3-N1, R4.L4-N1

The latch schematic contains three external nets, D, ENABLE, and Q. The symbol pins on the latch connect these nets to the hierarchical level mentioned above.

Automatic Aliasing of Nets

When a design is loaded into the Hierarchy Navigator, nets take the name of the highest (top-level) net in the design. That is, the name of top-level net propagates downward through the hierarchy to override the local name. By forcing all nets to the same name, this aliasing feature greatly speeds signal tracing in a multi-level design.

In the preceding example, the net name D from the latch is overridden by the higher-level external reference to become D1, D2, D3.... This override becomes

the reference at all levels of the hierarchy. If, in the suggested 16-bit register, the D0, D1, D2... inputs were connected to wires named and marked `Bit0`, `Bit1`, ... `Bit15`, these new names would take precedence and the D0, D1, D2... names would no longer be accessible at any level of the hierarchy.

Hierarchical Design Examples

ABEL-HDL Hierarchy Example

The first example below shows an upper-level ABEL-HDL module (`top.abl`) that references a lower-level ABEL-HDL module (`add.abl`). Following that, the example shows a lower-level module implemented as an ABEL-HDL block, while the figure shows the lower-level module implemented as a schematic block (`add.sch`). Both `add.abl` and `add.sch` can be instantiated in the upper-level source `top.abl`.

Top-level ABEL-HDL Module (`top.abl`)

```
MODULE top

"inputs
AIN,BIN,CARRYIN pin;

"outputs
CARRYOUT,SUMOUT pin;

add INTERFACE(A,B,CI -> SUM,CO);

my_add functional_block add;

EQUATIONS
my_add.A = AIN;
my_add.B = BIN;
my_add.CI = CARRYIN;
SUMOUT = my_add.SUM;
CARRYOUT = my_add.CO;

END
```


Lower-level ABEL-HDL module (add.abl)

```

MODULE add

"inputs
A,B,CI    pin;

"outputs
CO,SUM    pin;

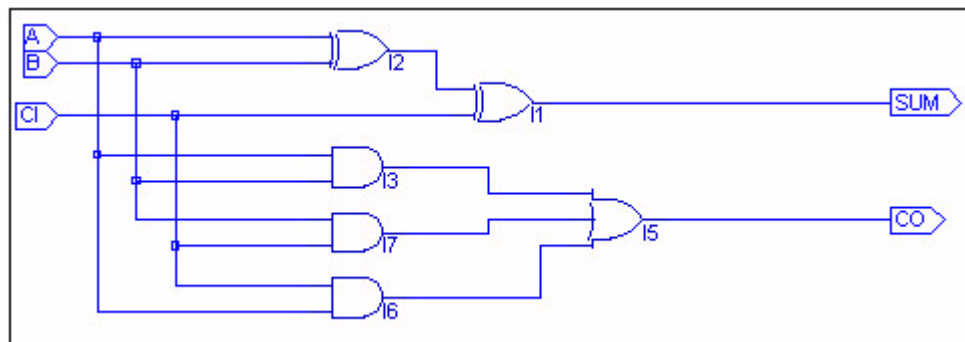
EQUATIONS

SUM =     A&B&CI
         +!A&!B&CI
         +!A&B&!CI
         +A&!B&!CI;

CO =     A&B
         +A&CI
         +B&CI;

END

```

Lower-level Schematic block (add.sch)

Note: If you are in a lower-level schematic, you can choose Add > New Block Symbol and then click Use Data From This Block on the dialog box to automatically create a functional block symbol for the current schematic.

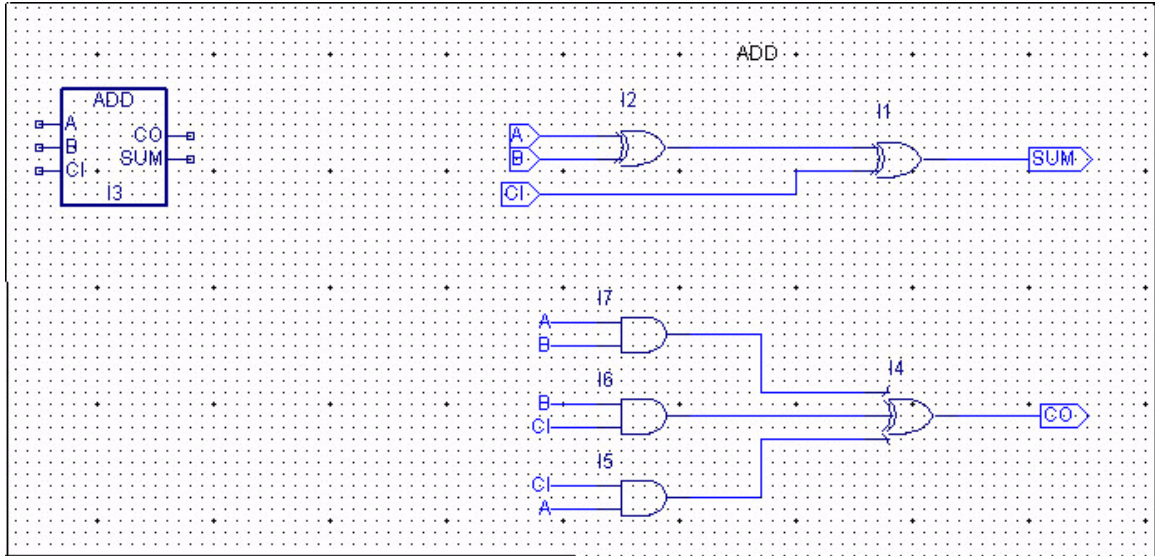
The name of the lower-level schematic must match the block name (schematic) or the interface name (ABEL-HDL) in the upper-level module. This associates the lower-level module with the symbol representing it. For example, the schematic in the Figure must be named add.sch.

The net name in the lower-level schematic corresponds to the pin names in the upper-level module that can be either schematic or ABEL-HDL.

Schematic Hierarchy Example

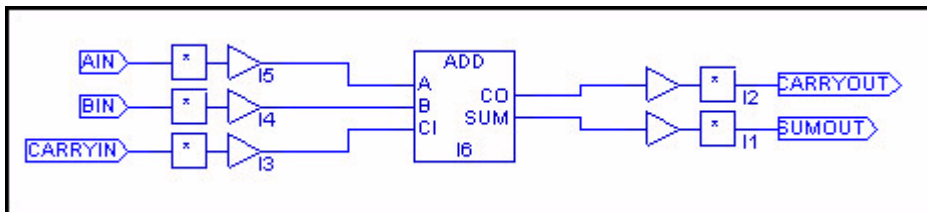
The figure below shows an example of how the new symbol corresponds to an underlying schematic. In this figure, pin A on the Block symbol corresponds to the net in the schematic, which is also named A. The other pins, B, CI (Carry In), CO (Carry Out) and SUM, also correspond to named nets in the schematic.

A block symbol and its underlying schematic



This following figure shows one top-level schematic and different ways to implement the lower-level modules.

Top-level Schematic for Top (top.sch)



Note: If you are in a lower-level schematic, you can choose Add > New Block Symbol and then click Use Data From This Block on the dialog box to automatically create a functional block symbol for the current schematic.

The name of the lower-level schematic must match the block name (schematic) or the interface name (ABEL-HDL) in the upper-level module. This associates the lower-level module with the symbol representing it. The above schematic must be named add.sch.

The net name in the lower-level schematic corresponds to the pin names in the upper-level module that can be either schematic or ABEL-HDL.

The symbol should be a Block symbol.

Lower-level ABEL-HDL Module for Add Block Symbol

```

MODULE add

"inputs
A,B,CI    pin;

"outputs
CO,SUM    pin;

EQUATIONS
SUM =      A&B&CI
          +!A&!B&CI
          +!A&B&!CI
          +A&!B&!CI;

CO =       A&B
          +A&CI
          +B&CI;

END

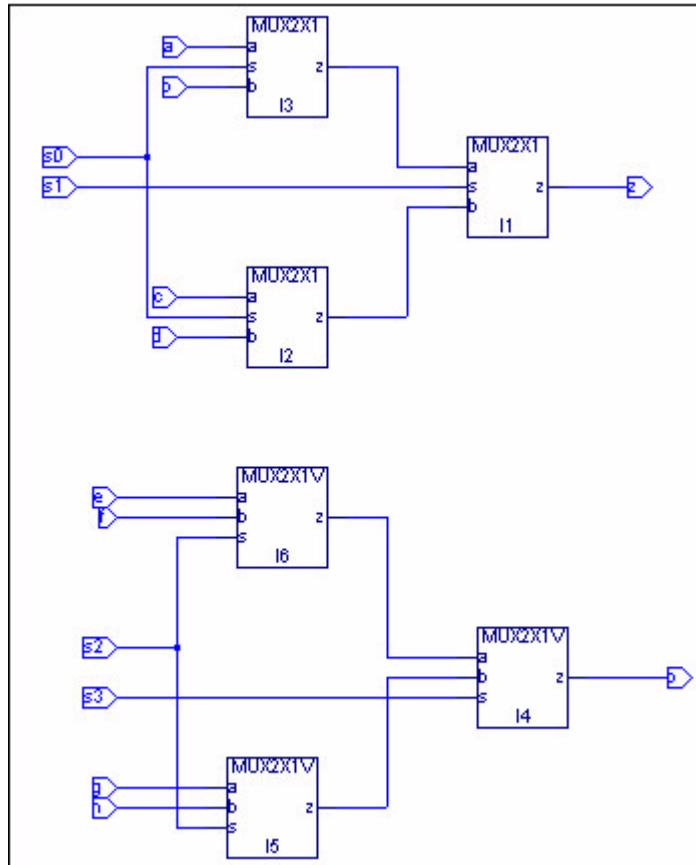
```

Note: *It is best to create the lowest-level sources first and then import or create the higher-level sources.*

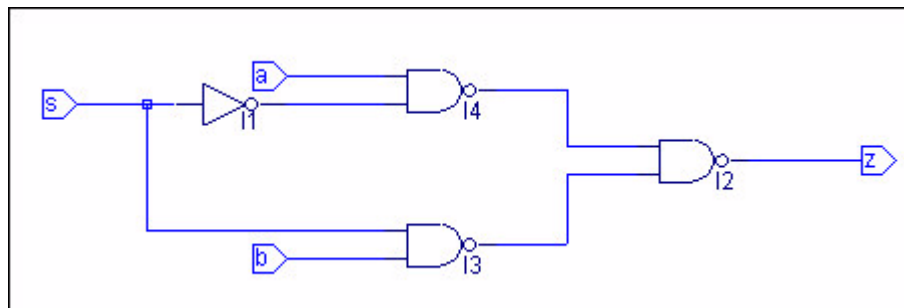
Schematic/Verilog HDL Hierarchy Example

The first figure shows the upper-level schematic `mux4x1.sch` that references a lower-level schematic and a lower-level Verilog HDL module. The second figure shows the lower-level schematic `mux2x1.sch`, and the file after the second figure shows the lower-level Verilog HDL module `mux2x1v.v`.

Top-level Schematic (mux4x1.sch)



Lower-level Schematic (mux2x1.sch)



Lower-level Verilog HDL (mux2x1v.v)

```
module mux2x1v(a,b,s, z);
```

```
output z;
```

```
input a, b, s;
```

```
reg z;
```

```

always @(a or b or s)
begin
  case (s)
    1'b1: z = b;
    1'b0: z = a;
    default: z = 'bx;
  endcase
end

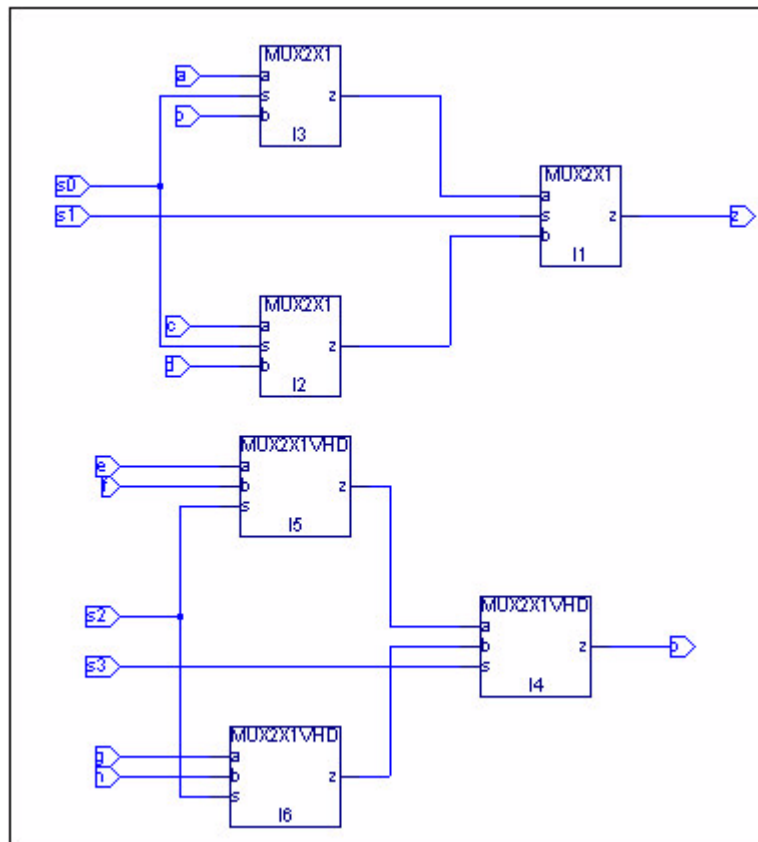
endmodule

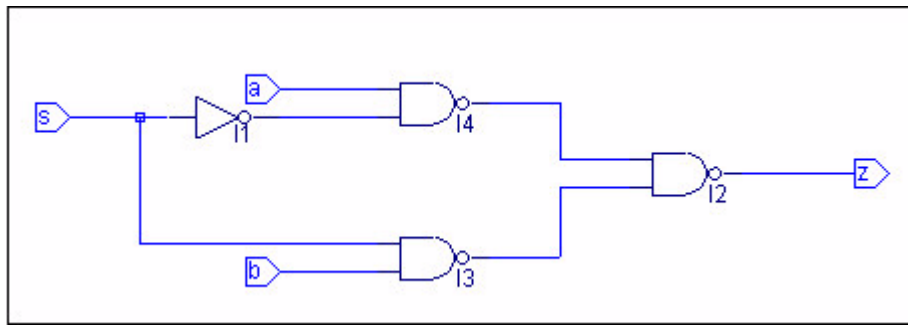
```

Schematic/VHDL Hierarchy Example

The first figure shows the upper-level schematic `mux4x1.sch` that references a lower-level schematic and a lower-level VHDL module. The second figure shows the lower-level schematic `mux2x1.sch`, and the file after the second figure shows the lower-level VHDL module `mux2x1vhd.vhd`.

Top-level Schematic (mux4x1.sch)



Lower-level Schematic (mux2x1.sch)**Lower-level VHDL Module (mux2x1vhd.vhd)**

```

library ieee;
use ieee.std_logic_1164.all;

entity mux2x1vhd is
    port ( z: out std_logic;
          a, b, s: in std_logic );
end;

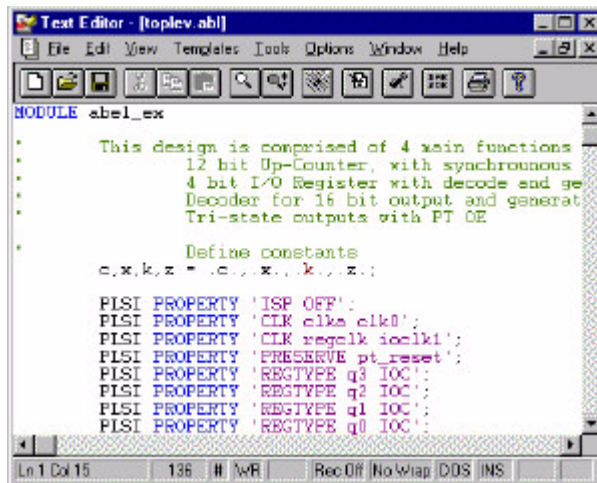
architecture mux2x1_arch of mux2x1vhd is
begin
    process (s, a, b)
    begin
        case s is
            when '0' =>
                z <= a;
            when '1' =>
                z <= b;
            when others =>
                z <= 'X';
        end case;
    end process;
end mux2x1_arch;

```

Design Entry

ABEL-HDL Design

ABEL-HDL is a hierarchical hardware description language that supports a variety of behavioral input forms, including high-level equations, state diagrams, and truth tables. The ispLEVER ABEL-HDL compiler and supporting software functionally verify ABEL-HDL designs through simulation. The compilers then implement the designs in a programmable IC. ABEL-HDL designs can also be transferred to other design environments through standard format, design transfer files.



```
Text Editor - [toplev.abl]
File Edit View Templates Tools Options Window Help
MODULE abel_ex
    This design is comprised of 4 main functions
    12 bit Up-Counter, with synchronous
    4 bit I/O Register with decode and gen
    Decoder for 16 bit output and generat
    Tri-state outputs with PT OE
    Define constants
    c,x,k,z = .c.,.x.,.k.,.z.;
    PLSI PROPERTY 'ISP OFF';
    PLSI PROPERTY 'CLK clk clk';
    PLSI PROPERTY 'CLK negclk negclk';
    PLSI PROPERTY 'PRESERVE pt_reset';
    PLSI PROPERTY 'REGTYPE q3 IOC';
    PLSI PROPERTY 'REGTYPE q2 IOC';
    PLSI PROPERTY 'REGTYPE q1 IOC';
    PLSI PROPERTY 'REGTYPE q0 IOC';
```

Using a Template to Create an ABEL-HDL Source

This procedure describes how to enter an ABEL-HDL design description using a template to create a single ABEL-HDL source.

To create a template for an ABEL-HDL source file:

1. In the Project Navigator, choose **Source > New** to open the New Source dialog box.
2. Select **ABEL-HDL Module** and click **OK**. The Text Editor opens, and a dialog box prompts you for a module name, file name, and title.
3. Type a Module Name, which is the name of the MODULE statement. The MODULE statement is required. It defines the beginning of the module and must be paired with an END statement. The MODULE statement also indicates whether any module arguments are used.
4. Type a File Name. The file extension can be omitted.

Note: *The module name and file name should have the same base name. (The base name is the name without the 3-character extension.) If the module and file names are different, some automatic functions in the Project Navigator might fail to run properly.*

5. Optionally, type a descriptive Title for the module.
6. When you have finished typing the information, click **OK**. You now have a template for an ABEL-HDL source file.

Entering Declarations

The Declarations section specifies the names and attributes of signals used in the design; defines constants, macros, and states; declares lower-level modules and schematics; and optionally declares a device. Each module must have at least one DECLARATIONS section, and declarations affect only the module in which they are defined. If a TITLE statement exists in the template file, enter these statements after the TITLE statement.

Using the *andff* module above as an example, the following describes the DECLARATIONS statement for the three inputs (two AND gate inputs and the clock) and the output.

DECLARATIONS

```
input_1, input_2, Clk      pin;
output_q                  pin istype 'reg';
```

These two statements declare four signals (*input_1*, *input_2*, *Clk*, and *output_q*).

Note: *ABEL-HDL does not have an explicit declaration for inputs and outputs. Whether a given signal is an input or an output depends on how it is used in the design description that follows. The signal *output_q* is declared to be type 'reg', which implies that it is a registered output pin. The actual behavior of *output_q*, however, is specified using one or more equations.*

Entering Logic Descriptions

You can use Equations, a State diagram, or a Truth table to describe your logic design. The following EQUATIONS statement describes the actual behavior of the *andff* example module.

EQUATIONS

```
output_q      := input_1 & input_2;
output_q.clk  = Clk;
```

These two equations define the data to be loaded on the registered output, and define the clocking function for the output.

Entering Test Vectors

The traditional method for testing ABEL-HDL designs is to use test vectors. Test vectors are sets of input stimulus values and corresponding expected outputs that can be used with both Equation and JEDEC simulators.

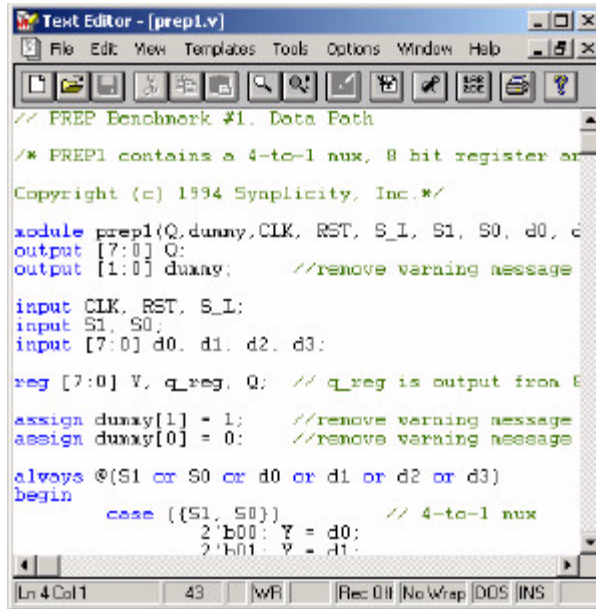
You can specify test vectors in two ways: in the ABEL-HDL source file, or in an external ABEL Test Vector File (.abv). When you specify the test vectors in the ABEL-HDL source file, the ispLEVER software creates a "dummy" ABV file that points to the ABEL-HDL source containing the vectors. This file is necessary because an ABV file is required in order to have access to the Equation and JEDEC simulation processes.

To continue with the example *andff* module, the following describes the TEST_VECTORS statement.

```
TEST_VECTORS
    ([Clk, input_1 , input_2] -> output_q)
    [ 0 ,    0    ,    0    ] ->    0;
    [.C.,    0    ,    0    ] ->    0;
    [.C.,    0    ,    1    ] ->    0;
    [.C.,    1    ,    1    ] ->    1;
```

Verilog HDL Design

The ispLEVER software supports Verilog HDL, a hardware description language used to design and document electronic systems. Verilog HDL allows designers to design at various levels of abstraction.



```

Text Editor - [prep1.v]
File Edit View Templates Tools Options Window Help
// PREP Benchmark #1. Data Path
/* PREP1 contains a 4-to-1 mux, 8 bit register and
Copyright (c) 1994 Synplcity, Inc.*/

module prep1(Q,dummy,CLK, RST, S_L, S1, S0, d0, d1, d2, d3)
output [7:0] Q;
output [1:0] dummy; //remove warning message

input CLK, RST, S_L;
input S1, S0;
input [7:0] d0, d1, d2, d3;

reg [7:0] Y, q_reg, Q; // q_reg is output from E

assign dummy[1] = 1; //remove warning message
assign dummy[0] = 0; //remove warning message

always @(S1 or S0 or d0 or d1 or d2 or d3)
begin
    case ({S1, S0}) // 4-to-1 mux
        2'b00: Y = d0;
        2'b01: Y = d1;
    endcase
end

```

Adding a Verilog HDL Module to Your Design

To add a Verilog HDL module to a design, you can either import a .v file, or create a new Verilog HDL module file with the Text Editor.

Creating a New Verilog HDL Module

You can use the Text Editor to create a new Verilog HDL module.

To create a new Verilog HDL module:

1. In the Project Navigator, choose **Source > New** to open the New Source dialog box.
2. Select **Verilog Module** and click **OK**. The Text Editor window appears together with the New Verilog Module dialog box.
3. In the dialog box, type the relevant contents into the text fields.
4. Click **OK**. The new Verilog HDL file appears in the Text Editor window.
5. Use the commands on the Edit menu to Cut, Copy, Paste, Find, or Replace text.

Synthesizing Your Verilog HDL Design

The ispLEVER software provides two synthesis tools that are integrated into the Project Navigator environment: Synplcity Synplify and Exemplar LeonardoSpectrum. You can synthesize your Verilog HDL design as a stand-

alone process by choosing the synthesis tool from the Lattice Semiconductor Programs in your Start menu, or you can synthesize automatically and seamlessly within the Project Navigator.

In Project Navigator, select your synthesis tool in one of two ways:

- Choose **Options > Select RTL Synthesis** and make your selection.
- Choose **Tools** and make your selection.

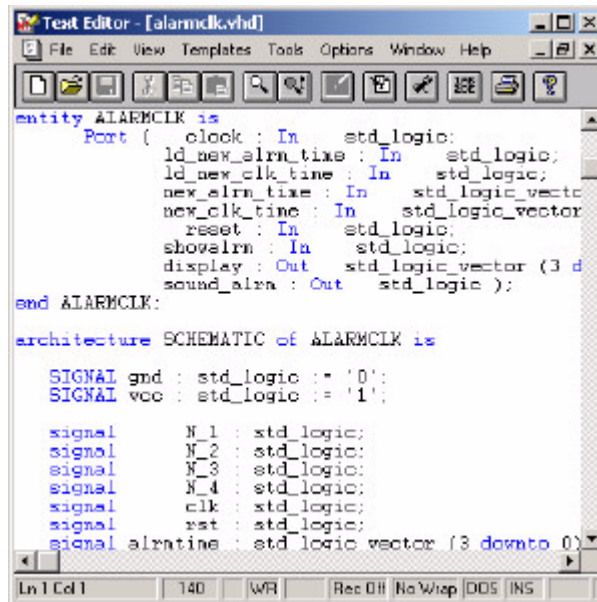
Synthesis then takes place automatically and seamlessly as you process your design.

VHDL Design

VHDL is a language for describing the structure and function of integrated circuits. VHDL allows you to:

- Describe the hierarchical structure and interconnect of a design.
- Specify the function of designs using familiar programming language forms.

Simulate the design before being manufactured, so that design alternatives can be quickly compared and tested.



```

entity ALARMCLK is
  Port (
    clock : In    std_logic;
    ld_new_alarm_time : In    std_logic;
    ld_new_clk_time : In    std_logic;
    new_alarm_time : In    std_logic_vector;
    new_clk_time : In    std_logic_vector;
    reset : In    std_logic;
    showalarm : In    std_logic;
    display : Out    std_logic_vector (3 d
    sound_alra : Out    std_logic );
end ALARMCLK;

architecture SCHEMATIC of ALARMCLK is

  SIGNAL gnd : std_logic := '0';
  SIGNAL vcc : std_logic := '1';

  signal    N_1 : std_logic;
  signal    N_2 : std_logic;
  signal    N_3 : std_logic;
  signal    N_4 : std_logic;
  signal    clk : std_logic;
  signal    rst : std_logic;
  signal    alrntime : std_logic_vector (3 downto 0);

```

Adding a VHDL Module to Your Design

To add a VHDL module to a design, you can either import a .vhd file, or create a new VHDL module file with the Text Editor.

Creating a New VHDL Module

You can use the Text Editor to create a new VHDL module.

To create a new VHDL module:

1. In the Project Navigator, choose **Source > New** to open the New Source dialog box.
2. In the dialog box, select VHDL Module and click **OK**. The Text Editor window appears together with the New VHDL Source dialog box.
3. In the New VHDL Source dialog box, type the relevant contents into the text fields.
4. Click **OK**. The new VHDL file appears in the Text Editor window.

5. Use the commands in the Edit menu to Cut, Copy, Paste, or Replace text.

Synthesizing your VHDL Design

The ispLEVER software provides two synthesis tools that are integrated into the Project Navigator environment: Synplicity Synplify and Exemplar LeonardoSpectrum. You can synthesize your VHDL design as a stand-alone process by choosing the synthesis tool from the Lattice Semiconductor Programs in your Start menu, or you can synthesize automatically and seamlessly within the Project Navigator.

In Project Navigator, select your synthesis tool in one of two ways:

- Choose **Options** > **Select RTL Synthesis** and make your selection.
- Choose **Tools** and make your selection.

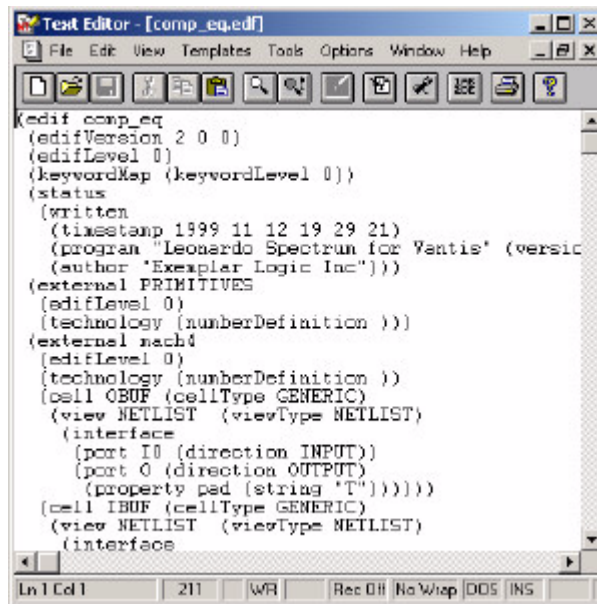
Synthesis then takes place automatically and seamlessly as you process your design.

EDIF Design

The Electronic Design Interchange Format (EDIF) is a format used to exchange design data between different ECAD systems.

The EDIF format is designed to be written and read by computer programs that are constituent parts of EDA systems or tools. Its syntax has been designed for easy machine parsing and is similar to LISP.

The ispLEVER software supports EDIF Version 2 0 0.



```

Kedif comp_eq
(edifVersion 2 0 0)
(edifLevel 0)
(keywordMap (keywordLevel 0))
(status
  (written
    (timestamp 1999 11 12 19 29 21)
    (program "Leonardo Spectrum for Vantis" (versic
      (author "Exemplar Logic Inc")))
  (external PRIMITIVES
    (edifLevel 0)
    (technology (numberDefinition ))
  (external mach4
    (edifLevel 0)
    (technology (numberDefinition ))
    (cell OBUF (cellType GENERIC)
      (view NETLIST (viewType NETLIST)
        (interface
          (port I0 (direction INPUT))
          (port O (direction OUTPUT)
            (property ped (string "T"))))))
    (cell IBUF (cellType GENERIC)
      (view NETLIST (viewType NETLIST)
        (interface
  
```

Importing an EDIF Netlist

You can import a design netlist description into the ispLEVER software from a third-party synthesis or schematic tool if the design file is formatted as EDIF 2 0 0.

To import an EDIF netlist into your project:

1. In the Project Navigator, choose **Source > Import** to open the Import File dialog box.
2. Change Files of type to **EDIF Netlist** (*.ed*), and then select the EDIF file that you want to import.
3. Click **Open** to open the Import EDIF dialog box.
4. The default setting for power and ground in the ispLEVER software are the VCC and GND symbols. If you know that the EDIF generated by other tools uses a different convention, you can change it in the window. Select **Custom**. Select either **Symbol** or **Net** representation. Then type the new names for VCC and GND.
5. If you are following the recommendation from Lattice for generating the EDIF file from the supported third party design kit, you can then select

CAE Vendors. Then from the list, you can choose the vendor that generated the EDIF file.

6. Click **OK**. The software adds the selected EDIF file (.edf) to the project sources.

Translating EDIF Properties

By default, the ispLEVER software ignores EDIF properties. If you want the ispLEVER software to translate EDIF properties to design constraints for the Fitter, do the following:

1. In Project Navigator, choose **Tools > Import Source Constraint Option** to open the dialog box. The Import Source Constraints Option dialog box lets you import constraints, such as Location (pin/node) Assignments, Group Assignments, and Output Slew Rate, from source files (ABEL, schematic, or EDIF).
2. In the dialog box, select Auto Import Source Constraints.
3. Click **OK**.

When you select this option, the ispLEVER software displays a confirmation dialog box prior to implementing the function. This confirmation dialog box appears every time you run the Fit Design process, unless you select the Do Not Import Source Constraints option.

On the warning message dialog box, if you click **Yes**, the constraints from the source files are written into the project constraint file.

Important: Constraints from source files and existing constraints in the project constraint file are not merged; existing constraints are overridden by the new constraints.

Existing constraints (only Location Assignments, Group Assignments, and Output Slew Rate are affected) in the project are cleared, regardless of constraints that might exist in the source file. If there are constraints in the source file, the new constraints are written into the project constraint file. If there are no constraints in the source file, no constraints are written into the file.

EDIF Properties

The ispLEVER software will take design-specific constraints from the properties in the EDIF netlist. The following is the list of properties that the Fitter supports.

PIN LOCATION Property

Name: LOC

Value: {PIN # }

Example: LOC = P20

Scope: IO PORT, net connect to the IO port.

GROUPING Property

Name: GROUPING

Value: GROUP NAME

Example: Use the following command to assign signal locations in your design. In this case, you have a list of internal node: a, b, and c, and you want to assign them into a group "mg." The location of this group needs to be Block "A", Segment "2":

On net a, grouping = mg

On net a, loc = "A, 2"

On net b, grouping = mg

On net b, loc = "A, 2"

On net c, grouping = mg

On net c, loc = "A, 2"

OUTPUT SLEW Property

Name: SLEW

Value: {Fast, Slow}

Example: To set port A to high slew, put the following property on the net or port:

SLEW=Fast

Scope: OUTPUT PORT/NET.

SIGNAL OPTIMIZATION Property

Name: OPT

Value: {KEEP, COLLAPSE}

Scope: On any net of the design.

OPEN DRAIN Property

Name: OPENDRAIN

Value: {On/Off}

Example: To set port A to an open drain, put the following property on the net or port:

OPENDRAIN=on

Scope: OUTPUT PORT/NET.

PULL Property

Name: PULL

Value: {On/Off/Hold}

Example: To set port A to pull up, put the following property on the net or port:

PULL=on

Scope: OUTPUT PORT/NET.

OUTPUT VOLTAGE Property

Name: VOLTAGE

Value: {VCC/VCCIO}

Example: To set port A to output voltage at VCCIO level, put the following property on the net or port:

VOLTAGE=VCCIO

Scope: OUTPUT PORT/NET.

Schematic Design

Introduction to Schematic Design

The schematic design entry environment is a set of tools that allows you to capture the structure of a design as either a flat description or a hierarchical set of components, and the connectivity between these components. Then you can use this description to drive the Fitter and verification tools. Designs can be single-level (flat) or multi-level (hierarchical). Schematics can be drawn on multiple "sheets" and be of any size.

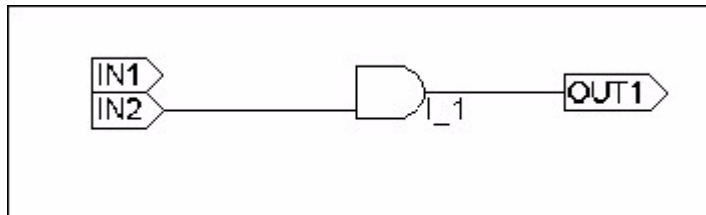
The Schematic Editor works in conjunction with the Hierarchy Navigator, Symbol Editor, and Library Manager programs.

Schematic Overview

Unless you're using them just for documentation, schematics are actually the starting point of the development process, not the goal. The schematic will eventually be used to analyze the device's behavior using the Functional Simulator and the Waveform Viewer.

The ispLEVER software includes a Schematic Editor for creating and editing schematic sources. The figure below is an example schematic that represents an AND gate.

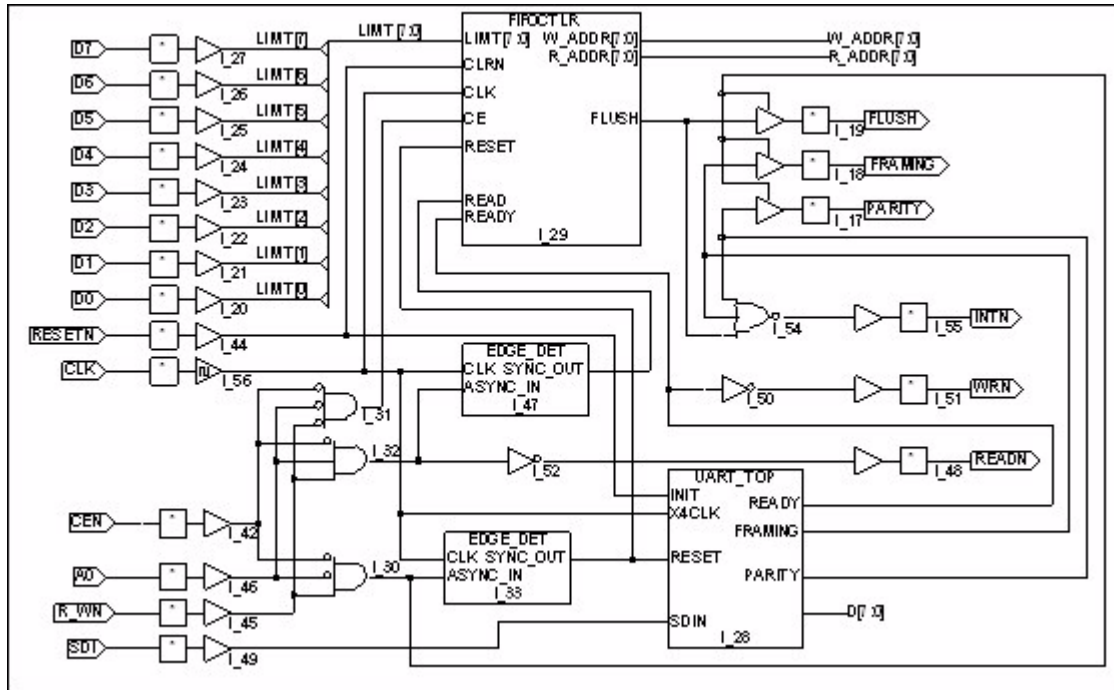
Schematic of an AND gate



In the above schematic there are two inputs (labeled **IN1** and **IN2**) and one output (**OUT1**). The function of **I_1** is to combine the signals **IN1** and **IN2**.

The following figure is a much more complex schematic that represents a programmable IC.

Complex schematic of a programmable IC



What is a Schematic?

The schematic file (saved as a `.sch` file) describes your circuit in terms of the components used and how they connect to each other. The schematic can be used to create netlists for the ispLEVER Fitter.

A schematic can represent a simple logic process (such as an AND gate) or a more complex component in your design (such as a Register). It can also represent the top-level of your design.

What do Schematics Consist of?

A schematic is composed of the following items:

Symbols - These can be symbols from the standard Symbol libraries, symbols representing other schematics you have drawn (Block symbols), or symbols you have created from scratch.

Wires - Wires connect the symbols. They can be single-signal (nets) or multiple-signal (buses).

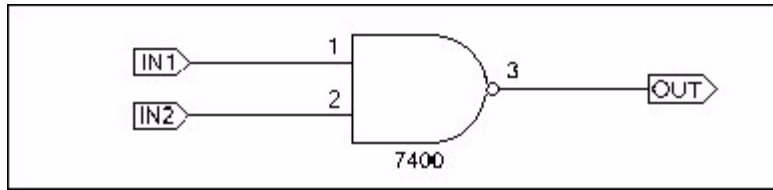
I/O Markers - I/O markers show where signals enter or exit the schematic and the direction (polarity) of the signal. That is, whether it is an input, output, or bi-directional signal.

Graphics & Text - Graphics and text are usually added to display explanatory data. They are optional and have no electrical meaning.

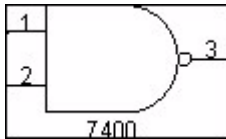
A valid schematic must contain at least the first three components—*symbols*, *wires*, and *I/O markers*. For instance, a single, isolated component symbol cannot be the only element in a schematic. The schematic must include I/O

markers for the external connections to the schematic, and these markers must be connected to the symbol with wires.

A valid schematic



Invalid Schematic (no wires or I/O markers)



Symbols

Symbols are graphic representations of components. The term "symbol" usually refers to an electrical symbol, such as a gate or a sub-circuit. You can draw graphic-only symbols (such as title blocks) with the Symbol Editor, but these have no electrical meaning.

Symbols are the most basic elements of a schematic. Symbols represent primitive design elements, whether those elements are complete gates or a complex macro. A symbol can also be the hierarchical representation of a sub-circuit, or a "Block" symbol.

Symbol Information

Each schematic symbol is a file ending with a `.sym` extension, and may be included in a library file with a `.lib` extension. The symbol file contains four types of information: *graphics*, *text*, *pins*, and *attributes*.

Graphics and Text

Graphics are pictures of the symbols. Symbol graphics have no electrical meaning and show only the position of the component in the circuit. A symbol's attributes and pins, not the graphics that represent it, define the electrical behavior of a symbol. Explanatory or descriptive text displayed with a symbol is also considered "graphic" information without electrical meaning.

Pins

Symbol pins are the connecting points between the symbol and the schematic wiring. If the symbol represents an individual component, the symbol pin represents the physical pin where a conductor can be attached. If the symbol represents a subcircuit (block symbol), the symbol pin represents a connection to an internal net of the subcircuit.

Attributes

Each symbol has a number of predefined attributes that describe its component type and other unchanging characteristics. (These were discussed briefly in the preceding section, "Symbols. "). Other attributes can be given values after the symbol is placed in the schematic.

Wires

Wires are the lines that electrically connect the symbol pins. Symbol pins are the only connection points for wires. You cannot connect wires to the symbol body itself.

There are two types of wires: single-wire *nets* and multiple-wire *buses*. Buses allow more than one signal to be routed as single line.

Wires (both nets and buses) are added to schematics using the **Add > Wire** or the **Add > Net Name** commands of the Schematic Editor.

Note: *There is only one kind of wire you can add (using the Add > Wire command). Whether it is a net or a bus depends on how you name the wire (using the Add > Net Name command). For instance, Buses are named as busname [numberlist] where busname is the name of the bus and numberlist is a list of numbers separated by commas representing each net in the bus. You can also draw and name a net or bus (you do not have to use the Add > Wire command) by choosing Add > Net Name and dragging on the schematic to draw the wire.*

Wire Names

Wires have names. These names identify the wires to the Schematic Editor and netlister programs.

You would normally name all wires that connect to inputs or outputs and any "internal" nets with signals you want to view during simulation. You can use any name you like, but you usually choose a name that suggests the name or function of the signal carried by that wire. If you don't give a wire a name, the Schematic Editor automatically supplies one, of the form N_*n* (where *n* is an integer).

Giving a single wire a compound name creates multi-wire buses. You can then tap off any signal you want anywhere along the bus.

Buses are most often used to group related signals, such as a 16-bit data path. However, a bus can be any combination of signals, related or not. Buses are especially useful when you need to route a large number of signals from one side of the schematic to the other.

Buses also make it possible for a single I/O marker to connect more than one signal to a Block symbol. The signal names don't have to match, but both pins must carry the same number of signals.

Net Attributes

Like symbols and symbol pins, nets (the wiring that connects symbols to each other and makes external connections) can also have attributes. These attributes

include the net's name, as assigned in the schematic. There are also net attributes that pass parameters to other programs, such as simulators.

I/O Markers

I/O markers mark the points at which signals leave or enter the schematic. They are required. Any unconnected wire without an I/O marker will eventually be flagged as an error when you try to create a netlist, run the simulator, or load the design into the Hierarchy Navigator.

I/O markers are added to schematics using the **Add > I/O Marker** command in the Schematic Editor.

The I/O marker automatically takes the name of the wire it is attached to. If the wire is a bus, the marker will have the same compound name as the bus.

When a Block symbol and its matching schematic are created, the I/O markers for the signals that enter and leave the schematic must have the same names as the corresponding pins on the Block symbol. The matching names identify which signal attaches to which pin.

Graphics

Although symbols, wires, and I/O markers are visible, graphical items, they also have a functional or electrical meaning. In this context, "graphics" refers to the non-functional graphical parts of the schematic.

For example, you might add graphics showing the expected waveforms at different points in the circuit. Or, you could draw the company's logo and add it to each schematic for identification.

The most common use of graphics is to create a title block. The block shows the name and address of your company, and can include the company logo and blank spaces for the project name, schematic sheet number, and so on.

The title block is a symbol (usually called **Title** and located in the project directory, in the `..\isptools\ispcpld\sym_libs\title.lib` library, or as the `..\isptools\ispcpld\generic\generic\misc\title.sym` file.). You can modify this symbol in the Symbol Editor and save the file (with the same name) in the project directory, or you can use the Library Manager to add the revised symbol to the `misc.lib` symbol library file.

Text

Text, like graphics, can provide additional information about the schematic or its project. Text can be placed anywhere on a schematic, even if it overlaps symbols or wires.

Text is added to schematics using the **Schematic Editor: Add > Text** command.

Naming Schematic and Symbol Files

When you name schematic or symbol files, observe the following rules:

- Observe DOS file naming conventions. The dollar sign (\$) cannot be the first letter of a file name, however.

- Don't type a file name extension; the Editor will add it automatically. If you enter a non-standard extension, the Editor will replace it with the correct extension (.sch for schematics, .sym for symbols, and .lib for symbol libraries).

Schematic Attributes

You use attributes to describe the characteristics or properties belonging to, or associated with, a **symbol**, **pin**, or **net**. Attributes only apply to describing characteristics in schematics. ABEL-HDL source files have their own syntax for describing characteristics.

Note: See each device family application note for a list of supported attributes.

Attribute Use

Attributes are primarily used to control certain aspects of design optimization and signal placement. You can also use attributes to control how the Fitter implements your design.

Attribute Types

There are two types of attributes used in the Schematic Editor: *symbol* and *net*.

Symbol attributes describe features related to a whole symbol. Symbol attributes usually apply only to the symbol on which they appear.

Net attributes describe characteristics associated with nets.

Attribute Components

Every attribute consists of four components: *name*, *value*, *modifier*, and *window*. The following is a brief discussion of attribute components as they apply to programmable IC design.

Attribute Name

An attribute's name identifies the attribute to the user. Width, Length, Refname, and PinNumber are examples of attribute names. Consult your **Vendor Kit** (Device Kit or Interface Kit) documentation for the names and descriptions of any attributes you need to use.

Attribute Value

An attribute can be assigned a *value*. A value is usually a number or a text string.

Attribute Modifier

An attribute modifier specifies the conditions under which an attribute's value can be modified. The attribute modifiers are grouped based on where you can edit their values:

- Anywhere in Design (<blank>)
- Not Editable (!)
- Symbol Only (-)

- Symbol or Schematic (\$)
- Derived (*)

Attribute Window

Attribute values are displayed in attribute windows. Attribute values cannot be displayed unless the symbol has at least one attribute window.

You add attribute windows to a symbol when you define the symbol. Each window is assigned a unique number and the default attribute that will be displayed in that window. (The window number *does not* have to match the number of the assigned attribute.) When the symbol is placed in a schematic, the value of the assigned attribute appears in the window.

You can temporarily change which attribute is displayed in an attribute window, using the Attribute Display command from the Options menu. This is useful when you need to view attributes that are not currently displayed.

Attribute windows in schematics can be repositioned, one at a time, with the Attribute Location command on the Edit menu. Repositioning can make a crowded schematic more readable.

Note: *Attribute windows do not have visible outlines. Rather, they are predefined areas on or near the symbol.*

Setting Attribute Values

The Schematic Editor passes attribute information to the Fitter. For those attributes that can be edited, you can set or override the values as follows:

- You can set **symbol** and **pin attribute** values for all occurrences of a symbol in the Symbol Editor.
- You can override the attribute values in any schematic where the symbol appears by using the Schematic Editor.
- You can override the attribute values in any design where the symbol appears by using the Hierarchy Navigator.

Default Values

Attribute values in a symbol definition become the default values for each symbol instance. These values are frequently overridden in the completed design, usually to optimize its performance.

When you are in the Symbol Editor, any attribute values you set in the Symbol Editor will be used.

When you are in the Schematic Editor, any attribute values you set in the Schematic Editor will be used. If you did not specify attributes for the symbol in the Schematic Editor, the Schematic Editor will use the values set for the symbol in the Symbol Editor.

When you are in the Hierarchy Navigator, any attribute values you set in the Hierarchy Navigator will be used. If you did not specify attributes for the symbol in the Hierarchy Navigator, the Hierarchy Navigator will use the values

set for the symbol in the Schematic Editor. If you did not specify attributes for the symbol in the Hierarchy Navigator or the Schematic Editor, the Hierarchy Navigator will use the values set for the symbol in the Symbol Editor. (Note that most netlisters use the values you set in the Hierarchy Navigator.)

Attributes that apply to all instances of a symbol (such as the vendor part number and the pin polarity) are generally assigned values when the symbol is created. Attributes that apply to a single instance (such as the instance name) are assigned after a symbol has been placed in the design.

The symbol libraries supplied with the ispLEVER software have predefined values for all the attributes required by most simulators and netlisters.

Displaying Attribute Values on a Schematic

Attribute values are displayed in attribute windows. Before an attribute can be displayed on a schematic, an attribute window number must be assigned to the attribute in the Symbol Editor.

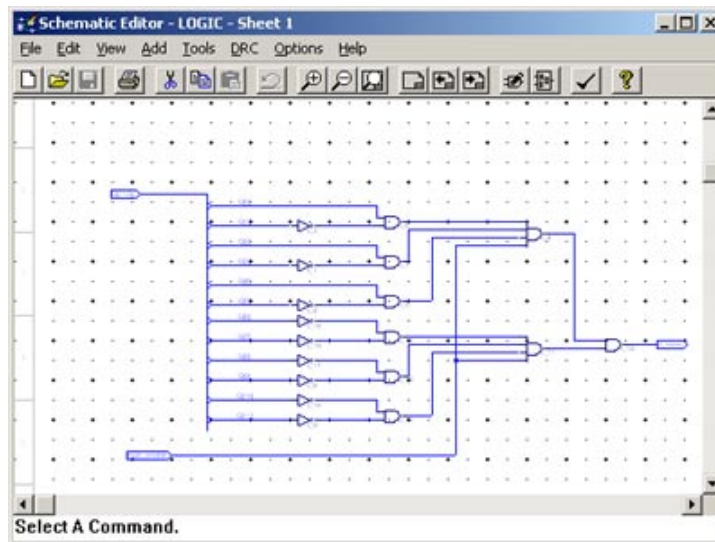
You can add an attribute window number to any value to display it. The attribute window number does not have to match the attribute number.

You can assign one attribute window number to several attributes. The attribute with the lowest attribute number *that has a value assigned* is displayed.

The Schematic Editor

You create schematic designs using the Schematic Editor. Schematics can be drawn on multiple "sheets" and be of any size.

The Schematic Editor can work in conjunction with the Hierarchy Navigator, Symbol Editor, and Library Manager programs.



Basic Schematic Editor Operation

The Schematic Editor uses an *action-object* command structure. That is, you select the action you want to perform (usually from a menu), and then you select the object you want to act on.

For example, to remove a symbol from a schematic, you first select the Delete command from the Edit menu. Then you point the mouse cursor at the symbol and click the mouse button.

Almost all commands remain in effect until you select a different command. For example, if the Add Wire command is active, you can continue to draw wires until you select a different command.

All commands that require another action or additional information will prompt you for it. The prompt line is at the lower-left corner of the Schematic Editor. Whenever you're not sure what to do, look at the prompt line.

The prompt line also displays what you type, such as the names of symbols and signals. You can edit this information as you type, using the ARROW, DEL, and BACKSPACE keys on your keyboard.

An error window immediately above the prompt line reports minor errors that prevent a command from completing its action. This prompt often explains how to fix the error. Major errors are reported in popup message box.

Error Recovery

The Schematic and Symbol Editors have a feature that increases the chance of fully recovering from a hardware or software malfunction (a "crash" or "lockup"). The first time you save a new design, the Schematic Editor creates a log file named `design._sc` (where `design` is the base name of the schematic file). The Symbol Editor log file is named `design._sy`.

The Editors check for a log file before opening an existing schematic or symbol. If the file exists, it means:

- Someone else on the network has this particular file currently open. (See the next section on network operation.)
- The last time the file was opened, the user did not exit the Editor normally.

When either Editor finds a log file, the Editor displays a warning message that asks you to verify that no one else is editing the target file. If someone else on the network is editing the file, you should not edit it, as you would overwrite the other person's changes. If you tell the Editor that someone else is editing the file, the Editor will not load it.

If you respond that no one else is using the file, the Editor assumes the log file is left from an interrupted editing session. The Editor then asks if you want to recover the file. If you respond **Yes**, the Editor uses the log file to recover any changes made to the schematic or symbol before the last editing session was interrupted.

The log file is not named until you first save a schematic. Be sure to save new schematics and symbols as soon as you start working on them, so that an easily identifiable log file will be available. (Until you save the file and name it, the Editor gives the log file an arbitrary name, such as `_SC60D5.TMP` in the directory specified by the `TMP` or `TEMP` environment variables.) Whenever you exit the Schematic or Symbol Editor normally, the Editor updates the schematic or symbol file (`design.sch` or `design.sym`) and erases the log file.

Network Operation

When schematics, symbol files, and other project components are shared on a network, some form of overwrite protection is required. The system does not allow two people to work on the same file at the same time.

The Schematic Editor uses the crash-recovery log file (described in the preceding section) to ensure single-user access. If you try to open a file and the Schematic Editor finds a log file for it, you are asked if someone else is using the file. If you answer **Yes**, you cannot access the file.

If you answer **No**, you will be allowed to open the file, even if someone else is editing it. Check with anyone who might be using the file before answering **No**.

Wiring the Schematic

Wires electrically connect schematic symbols. The symbol pins are the connection points for the wires. The Wire command is used to add wires between symbol pins. However, you can also use the Net Name and Bus Tap commands to add single wire segments.

Nets

Any single- or multi-wire connection between pins is called a *net* ("network"). The following topics explain how nets are named and how multi-wire nets (called *buses*) are created and named.

Net Names

The nets (networks) form the electrical connections among the components. Every net has a name, either assigned by you or by the Schematic Editor. Net names have two principal functions: *identification* and *interconnection*.

Note: *If your instance names use brackets [], parentheses (), or curly braces { }, the software replaces these characters with an underscore (_) because they are illegal characters in ABEL-HDL language.*

Meaningful net identifiers make a design easier to understand. Nets are usually given the names of the signals they carry.

If you don't assign a name, the Schematic Editor automatically assigns a unique name when you save the file, in the form of N_*nn*, where *nn* is an integer between 1 and 232 – 1 (4,294,967,295).

You can override any name assigned by the Schematic Editor and assign one of your own by using the Net Name command from the Add menu.

Interconnection

If a wire segment attached to a symbol pin is given the name of a net or bus, the pin is attached to that net or bus, even if you haven't drawn the connection on the schematic.

Two or more wires with no visible connection on the schematic are automatically connected if they have the same net name. Each wire is called a *branch* of that net. Inter-sheet connections are created in this way.

You can easily find implicit net connections with the Query command from the DRC menu. Click any net or bus. All wires with the same name are highlighted, on all sheets of the current schematic.

Nets with different names cannot be connected; the Schematic Editor will warn you if you try to "short" them.

Entering Net Names

Use the Net Name command from the Add menu to assign a name of your own choosing to a net. Your name replaces any name the Schematic Editor may already have assigned. If you assign the same name to two separate nets ("branches"), *they are connected*, even though no connection appears on the schematic. This feature makes it easy to connect widely spaced components without having to draw long wires across the schematic.

Net names you assign are always displayed; Editor-assigned names are not displayed. To avoid cluttering the schematic, you should name only those nets:

- That connect to other schematics.
- Whose functions need documentation or clarification.

- Whose signals you want to view in the Hierarchy Navigator.
- Whose signals you want to reference in simulation or timing analysis.
- Whose signals you want to identify in the Fitter report.

Placing Net Names

Once a net name (or group of names) is attached to the cursor, there are three ways to place it.

- **Click an empty space** to place the net name at an empty point on a sheet. This will be an error unless a net is eventually connected to the name flag.
- **Click a net** to place the net name on the selected net. If you click at the end of a net, the net name extends from the end. If you click the middle of a net, the net name is centered just above the net.
- **Drag to create the net** and place the net name on the newly created net. The net name and a single net segment can be placed *simultaneously*. You can then place subsequent wires and names by clicking a pin.

Note 1: *You can only define a horizontal or vertical net segment in this way. If either end of the segment connects to a perpendicular net or bus, a bus tap is created at that end of the segment. If the net was not a bus, it is promoted to one.*

Note 2: *The position of the name is determined by the segment ends at the time of placement. If both ends are connected, the name is placed in the middle. If neither end of the segment is connected, the name is placed at the starting point. If only one end of the segment is connected, the name is placed at the unconnected end.*

Note 3: *After you drag the mouse to or from a pin, you can place subsequent net names by clicking a pin. You don't need to drag. A wire segment is automatically added, of the same length as the one previously dragged. The name is attached as described above.*

Legal Characters in Net Names

The following characters can be used in net names:

A–Z, a–z, 0–9 All alphanumeric characters. Case is not significant.

' Apostrophe (single quote)

_ Underscore

Reserved Names

If B is the first character of a net name, the underscore cannot follow it as the second character (as in B_). The underscore cannot be used in a net name of the form "N_nn" (where "nn" is any integer). These names are reserved by the Schematic Editor for nets that have not been named by the user.

Logical Inversion

When either the apostrophe or underscore is the first character of a net name, the Schematic Editor draws the name with an overbar. Overbars are often used to suggest logical inversion. The apostrophe or underscore is kept in the name and appears in the netlist, but it is not displayed.

Specifying Signal Direction

An I/O marker is a special indicator that identifies a net name as a device input, output, or bidirectional signal. This establishes *net polarity* (direction of signal flow) and indicates that the net is externally accessible.

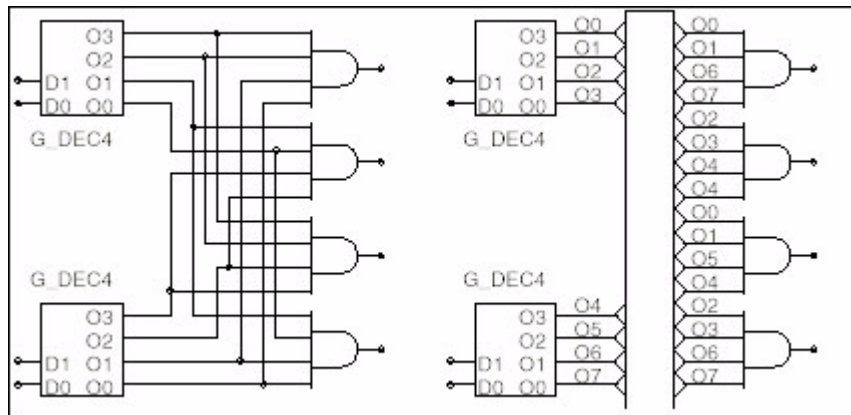
The Schematic Editor Consistency Check command uses I/O markers to flag any discrepancies in the polarity of marked signals and the symbol pins. Discrepancies in polarity are also flagged each time you run the Hierarchy Navigator.

Buses

A *bus* combines two or more signals into a single wire. Buses are a convenient way to group related signals. This grouping can produce a less cluttered, functionally clearer drawing and clarify the connection between the main circuit and a Block symbol.

The figure below shows how a circuit appears before and after a bus has replaced individual wires. The two schematics are electrically equivalent.

Circuit before and after a bus has replaced individual wires



Ordered Buses

An ordered bus has a compound name consisting of the names of the signals that comprise the bus. Any signals can be combined into an ordered bus, whether or not they are related.

A net becomes an ordered bus when it is given a compound name. The net is promoted to an ordered bus containing the nets listed in the compound name. (The net is redrawn at twice its regular thickness to indicate that it's now a bus.)

A compound name is a list of two or more net names, separated by commas. For example:

```
READ, WRITE, MYNAME
```

represents the three signals READ, WRITE, and MYNAME. Spaces in compound names are ignored.

Adding a sequence of numbers to a name can also form a compound name. The sequence is specified as a starting number, an ending number, and an optional increment (default = 1). The numbers are positive integers, and are delimited by commas (,), dashes (-), or colons (:). The sequence is enclosed in brackets [], parentheses (), or curly braces { }.

The following are examples of sequential compound names.

Sequential Name	Signals
DATA[0-7]	DATA[0] DATA[1] ... DATA[7]
ADDR(0,14,2)	ADDR(0) ADDR(2) ADDR(4) ... ADDR(14)
IO{4:23:3}	IO{4} IO{7} IO{10} ... IO{22}

If the increment is greater than one, the ending number will not appear in the sequence if it does not equal the starting number plus an integral multiple of the increment (as in the third example above).

A compound name can also combine individual names and compound names in any order.

Sequential Name	Signals
CS,DATA{0:7},WR	CS DATA{0} DATA{1} ... DATA{7} WR

The order of the signals in the bus is the same as the order in which they are specified. The order is significant only when the bus is connected to a bus pin. (Bus pins are described in a later section, "Bus Pins.")

Note: If your instance names use brackets [], parentheses (), or curly braces { }, the software replaces these characters with an underscore (_) because they are illegal characters in ABEL-HDL language.

Unordered Buses

An unordered bus is nothing more than an unnamed wire with *bus taps*. A net with a single name (or any unnamed wire) is promoted to an unordered bus by attaching one or more bus taps to it. The order of the signals within an unordered bus is not defined and has no significance.

Although the order of the signals in an unordered bus has no significance, *you must name the wires connecting to the bus taps*, because the Schematic Editor would otherwise have no way of determining which symbol pin at one end connects to which symbol pin at the other end

Unordered buses provide a convenient way to route signals through the schematic with a minimum of visual clutter. They have no other function.

Unordered buses cannot connect to bus pins, because bus pins represent an ordered sequence of signals.

Bus Taps

Signals enter (or exit) a bus at points called *bus taps*. A bus tap can be added to any existing bus, net, or wire. If a net or wire is not already a bus, adding the tap automatically promotes it to a bus.

Naming the Tap

Once the tap has been added, choose the **Add > Net Name** command to name it. If the tap is from an ordered bus, the tap's name must match the name of a signal in the bus. If it does not, the Schematic Editor or Hierarchy Navigator will flag it as an error.

Note: *Wires entering and leaving any bus (ordered or unordered) must be tagged with a net name to indicate which signal is being tapped. Unnamed taps will eventually be flagged as errors.*

Connecting to Pins

A tapped signal connects to an ordinary symbol pin in the usual way. An ordered bus connects to a *bus pin* (a pin with multiple connections) directly. No taps are needed; the connections are made automatically. The first signal in the bus connects to the first signal in the bus pin, the second to the second, and so forth. Both the bus and the bus pin must contain the same number of signals.

Bus Pins

A *pin* represents either a physical pin on a real component or a signal from a lower-level schematic.

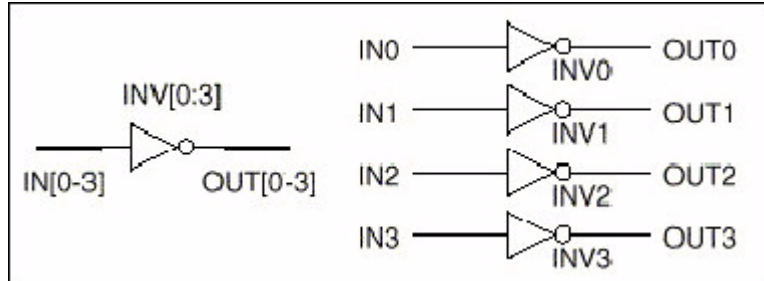
A *bus pin* represents a group of pins or signals. You create a bus pin by giving a pin a compound name. That is, a list of signals. If the pin connects to a Block symbol, each of the signals listed in the bus pin name must also appear in the schematic. This defines the connection between the Block symbol and its underlying schematic.

Ordered buses can connect directly to bus pins. The number of bits or signals attached to the bus must match the number of bits or signals attached to the bus pin.

The first signal in the bus (by definition, the first signal in the bus's name) is connected to the first signal represented by the pin. The remaining signals in the bus are connected to the remaining pins in the same order you assigned the signal names to the pins

Nets on Iterated Instances

Iterated instances allow a single symbol instance to represent multiple instances in a parallel connection. The figure below shows two ways of representing four parallel buffers. On the right, four separate inverters are added to the schematic. On the left, one symbol with the instance name of INV[0:3] represents the bank of four inverters.



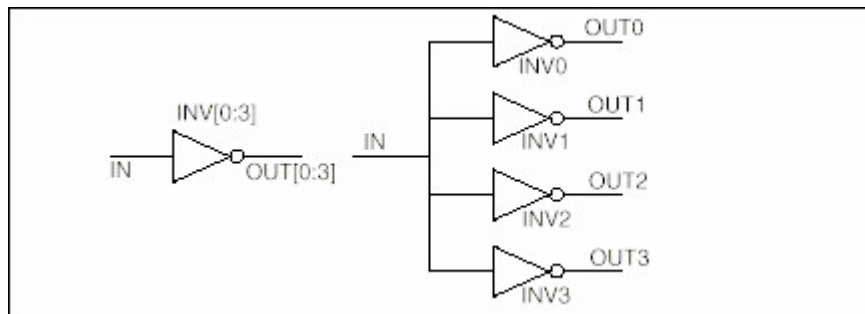
Compound Names

The input and output nets of an iterated instance can be given either single names or compound names. If the inputs or outputs are given a compound name, their nets are promoted to buses in which each instance's input or output is a separate signal.

Iterated buses work like any other bus. You can attach a bus (with the same number of signals) directly to them, as you would any other bus.

Single Names

If an iterated input or output is given a single net name, there is only one input or output net, and all the inputs or outputs connect in parallel to that single net. In the figure below, the input net is given a single name and the inputs of all four gates are connected in parallel to the net.



This feature is used most often on inputs, not outputs. Paralleled outputs represent a "wired-OR" configuration, which is usually drawn as separate gates, rather than as an iterated instance.

Bus and Net Connections to Iterated Instance

You can make an iterated instance of any symbol. A simple (non-bus) pin on an iterated instance remains a simple pin. The iteration does not convert the pin to a bus pin. The same rules of connection to a simple pin still apply.

Connections to the nets of iterated instances are made according to the following rules.

Rules for connection to nets of iterated instances

Connection	Rule
Simple net to simple pin	The net connects to corresponding pin on each instance.
Bus to bus pin	Successive bus signals connect to successive bus pins on successive instances. The bus and bus pin must have the same number of nets.
Simple net to bus pin	Not permitted. Only a bus can connect to a bus pin.
Bus to simple pin	The Nth signal of the bus connects to the scalar pin of the Nth instance. The width of the bus and the number of instances must match.

Wiring Constraints

The Schematic Editor enforces a number of wiring constraints. Most are intended to encourage clean layout and prevent ambiguous wiring patterns.

- All wire segments must end on Primary grid points.
- Wire segments must be oriented on the 45° and 90° axes.
- Wire segments cannot form acute angles. This applies to both crossing and connected wire segments.
- A maximum of two diagonal wire segments can connect at a point. A third vertical or horizontal segment can connect if it does not form an acute angle with either of the other segments.
- A maximum of three wires can connect at a pin or I/O marker.
- A net can have only one name (simple or compound).
- Two I/O markers with different net names cannot be connected by a wire.
- An I/O marker can connect only to a wire segment, never a pin. (Add a wire segment if you want a marker to be near a pin.)
- An I/O marker cannot be placed in the middle of a diagonal line, only at the end.
- An I/O marker cannot be placed at the crossing point of two wires, even if the wires are connected.
- A tap can only be placed on a vertical or horizontal section of a wire.
- Only one bus tap can be made at any point on a bus.

- A bus can contain only individual signals, not other buses. Attempting to give a net in a bus either a compound name or the name of another bus is flagged as an error.
- The relationship between an ordered bus (that is, a bus with a compound name) and the signals in that bus is strictly enforced. Naming the bus or one of its signals in a way that breaks this relationship is not permitted. For example, you cannot assign a bus tap a name that is not in the bus.

Debugging and Verifying a Schematic

The Schematic Editor has two levels of checking that report or prevent errors early in the design process.

First level errors are detected as you enter your schematic. For example, the Schematic Editor will not let you draw an isolated wire that forms a closed loop without connecting to anything else. It won't let you short together nets with different names.

Second level errors are recognized in the context of a complete design. An unconnected wire or pin, or an unnamed signal tapped from a bus, are normal during the first stages of a design. Some potential errors are always indicated, such as the dots on open pins and hanging line ends. Otherwise, errors of this type are reported only when schematics and symbols are combined in the design hierarchy.

You can check for errors and potential errors at any time.

"Unconnected Pin" Message

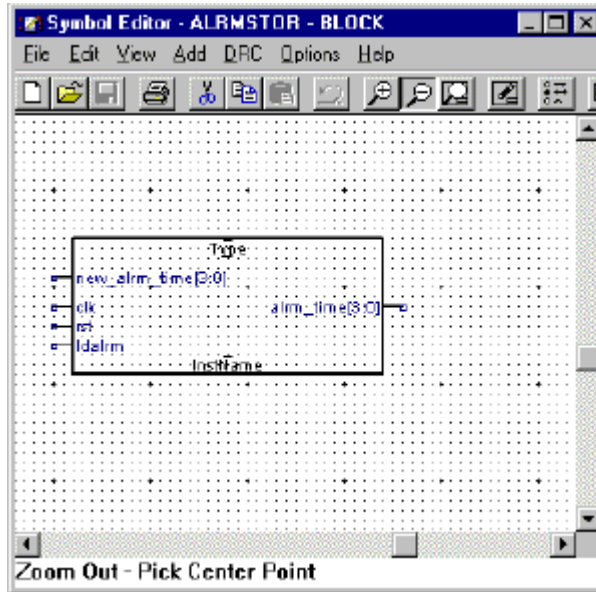
You can tell the Schematic Editor Consistency Check command to ignore intentionally unconnected pins by appropriately setting one of the pin's attributes. This attribute is named **OpenOK**. Use the Schematic Environment dialog box to add this attribute and set its Modify Option to "+", Assign in Schematic.

The attribute can be set in either the Schematic Editor or the Symbol Editor. If it is set in the Symbol Editor, the pin is never flagged as unconnected. The attribute can also be selectively set in the Schematic Editor to disable the message on specific pins, but not on all instances of the symbol.

Any value entered will inhibit the "Unconnected" message for that pin. A value of Yes, OK, or True is suggested.

The Symbol Editor

The Symbol Editor lets you construct schematic symbols. Besides the various lines, arcs, and boxes needed to create the symbol, you can also add text to give information about the symbol and its relationship with the rest of the circuit.



Symbol Elements

Schematics are constructed from symbols. A symbol can represent any logic component. Symbols are connected with wires (in the Schematic Editor) to create a complete schematic whose behavior can be verified and simulated.

A symbol is a picture; it has no inherent electrical meaning. Its electrical characteristics are supplied by *attributes* that describe the symbol's behavior. (The behavior of a Block symbol is described by the schematic file associated with that Block symbol.)

The ispLEVER software supplies you with an extensive set of symbols. You can also use the Symbol Editor to create Block symbols that represent a complete schematic, or part of one.

Symbols are composed of *graphic*, *pin*, and *attribute* elements.

Graphic elements are the picture of the symbol. They have no electrical meaning; they show only the location of the component in the schematic.

Pin elements on a symbol are points where a wire can be attached. The pins and wires are connected between symbols to circuit elements.

Buses cannot be connected to pins unless the pin is a bus pin. Only ordered buses can be connected to bus pins.

Attribute elements are properties of a symbol, pin, or net. Attributes can describe how the Fitter will optimize the symbol, or where it is placed in the device.

Positioning Master Symbols

Master symbols cannot be freely placed on a schematic sheet. Instead, they are automatically positioned at one of the corners. This permits resizing the sheet without having to move the title block or other annotations.

The sheet corner is determined by the location of the symbol's origin. If the origin is placed at the upper-right corner of the Master symbol, for example, the symbol will be positioned at the upper-right corner of the sheet.

Master symbols do not have pins.

Using Grids to Position Symbol Elements

All symbol elements are positioned on a grid. The default spacing of the grid is one-tenth of an inch (or 2.5 mm). (This spacing is set in the Graphic Options dialog box.)

Graphics are usually placed on the Primary grid, but you can align them with a Secondary grid that has two or four times the resolution. This finer resolution gives more precise control over the position of names, annotations, and graphic embellishments. The Options > Graphic Options command determines whether alignment is with the Primary or Secondary grids.

Symbol dimensions are stored as multiples of the Secondary grid units, *not* as absolute lengths. For example, if you redefine the Primary grid to be 0.2" (when it was previously 0.1"), symbol drawings and schematics will print out at twice their previous size.

Positioning Pins

Although graphics and text can be positioned at any of the three grid spacing, pins *must* be aligned with the Primary grid. Wires are drawn only on the Primary grid. If the pins are not on the Primary grid, you will not be able to attach wires to them.

The size of the drawing area can be increased with the Expand Page command from the Edit menu. Each time you choose the Expand Page command, the drawing area increases by 20 Primary grid units in each direction. The maximum size of the drawing area is 400 x 400 Primary grid units.

Selecting a Line Weight

Graphic objects can be drawn in two line weights: *normal* and *wide*.

Normal lines (default) are the same width as the wires in a schematic. You might use this weight for drawing all elements.

Wide lines are twice the width of Normal lines, the same weight as schematic buses. For example, you might use this weight to illustrate a bus pin connection.

Drawing Lines

When clicking to place the end points, lines are constrained to three principal directions: vertical, horizontal, and 45°. When you drag the line, the line can be at any angle as long as the end points fall on the grid being used. Switch to a finer grid to make it easier to place lines exactly where you want them.

Drawing Rectangles

Many symbols are based on a rectangular body. For non-rectangular symbols such as inverters or multiplexers, use the Line command to draw the outline.

Drawing Circles and Arcs

You can place full circles with the Circle command and create portions of circles with the Arc command. Arcs are useful for the curved sections of NAND and NOR gates.

Drawing Negation Bubbles

Negation bubbles are graphical and have no electrical significance. (Adding a negation bubble to a symbol does not change its logic. You must modify the symbol's attributes or the underlying schematic file.) You can add small or large bubbles:

- Choose **Add > Bubble** to draw a bubble one-half the Primary grid unit in diameter
- Choose **Add > Big Bubble** to draw a bubble one Primary grid unit in diameter

With either command, a bubble is attached to the cursor. Click the desired position in the schematic to place a bubble.

Drawing Text

Text can be added anywhere in the drawing window. Typical uses of text include:

- Notes about the symbol
- Title blocks
- Cross references

Text Size and Justification

Fixed text (as opposed to text appearing in attribute windows) can be drawn in up to eight sizes. (In the Windows version of the ispLEVER software, eight sizes are available. In the UNIX/Motif version, three sizes are available.) Text can be left justified, right justified, or centered. The controls for font size and justification are in the Graphic Options dialog box. Defaults for these values can be changed by using the Schematic Environment dialog box (Project Navigator: **Options > Schematic Configuration**).

Preparing Symbols for Schematics

A symbol needs special *links* so that it can be recognized and placed in a schematic. These links consist of *pins*, *attributes*, *attribute windows*, and the *symbol origin*.

Pins

Symbol pins are connection points for wires. Pins on Gate, Component, and Cell symbols represent the connection points on the device (pins or pads). Pins

on a Block symbol represent connections from one level of the hierarchy to the level below. Because Graphic and Master symbols don't represent electrical components, you can't attach pins to them.

Pins are the only symbol elements restricted to locations on the Primary grid, since wires must begin and end on Primary grid points.

Adding Pins

Symbol pins correspond to I/O markers on the underlying schematic and connect the device represented by the symbol to the rest of the circuit.

Pins are usually attached to the symbol on short lines extending outward from the symbol's body. However, pins can be attached anywhere inside or outside the symbol, with or without connecting lines.

Pins are electrical elements and are therefore restricted to locations on major grid intersections. There can be only one pin at any location.

Bus Pins

A bus pin is used to connect a bus to a symbol. Naturally, a bus pin must have as many nets or signals as the bus that connects to the pin.

One way to create a bus pin is to give a pin a name of the form:

```
bus_name [index1-index2]
```

Where *bus_name* is the name of an internal bus, and *index1* and *index2* specify the range of signals you want to connect. For example, if you need to connect nine signals, *index1* could be 5 and *index2* could be 13.

Alternatively, a bus pin can be defined by giving it a *compound name*—a list of bus names separated with commas (,):

```
name1, clk, mux[0-3], toggle
```

Bus Pin Limitations

Bus pins are allowed only on Block, Cell, and Component symbols. When a bus pin is created on a Component symbol, the numbers of the physical pins must be specified in the symbol definition.

These pin numbers are a list of pins assigned to the pin attributes BusPin_A through BusPin_H. When assigning bus pins, the normal PinNumber pin attribute *must not* have an assigned value.

The pin list can be divided sequentially among the eight attributes. Each individual attribute can hold about 200 characters. The list is delimited with commas or spaces, and can specify sequences of pins in parentheses () or square brackets []. Examples are

```
BusPin_A = 1, 3, 5, (7:10)
           7 pins: 1, 3, 5, 7, 8, 9, 10
BusPin_B = A1 B[2:4] C1
           5 pins: A1, B2, B3, B4, C1
```

Attributes

A symbol and each of its pins can have attributes. An attribute has a name and a value.

Attribute names are represented in the symbol's file by an integer. The correspondence between attribute names and integers is defined with the Symbol Attributes and Pin Attributes tabs in the Schematic Environment dialog box (Project Navigator: **Options > Schematic Configuration**). This arrangement allows the internal numbers to remain constant while the attribute names change to accommodate local practice or language.

Attribute values assigned in a symbol definition become the default values for each symbol instance. These values are frequently overridden in the completed design.

Symbol Attributes

Symbol Attributes are characteristics or properties associated with a symbol. Examples of symbol attributes are *PartNum*, *InstName*, *Width*, and *Type*. The standard symbol attributes (numbered 0–99) are reserved. You can create symbol attributes (numbered 100–199) using the Schematic Environment dialog box (Project Navigator: **Options > Schematic Configuration**).

Any attribute that is not defined as having a fixed value can later be modified in the Schematic Editor or Hierarchy Navigator using the Attribute command. The procedure is the same as editing pin attributes, described in the preceding section.

Pin Attributes

Pin Attributes are characteristics or properties associated with a pin. *PinName*, *Polarity*, and *PinNumber* are examples of Pin Attributes. Pin attributes are created with the Schematic Environment dialog box (Project Navigator: **Options > Schematic Configuration**) and their values are modified in the Symbol Editor.

Attribute Windows

Attribute windows are predefined areas on or near a symbol or pin in which attribute values are displayed. Attribute windows do not have a visible outline. If no value is displayed, there is no indication that an attribute window has been defined.

Numbers identify attribute windows. The association between an attribute window and an attribute is defined using the Symbol Attributes tab in the Schematic Environment dialog box (Project Navigator: **Options > Schematic Configuration**). An attribute window can have any number; it does not have to match the number of the attribute itself.

When a symbol is rotated or mirrored, the text in attribute windows retains its original position. This keeps it readable.

Setting Symbol Origins

When a symbol is placed in the Schematic Editor, the symbol is attached to the cursor. The point on the symbol attached to the cursor is called the *origin* of the symbol.

A newly created symbol has no origin. When the symbol is saved, the origin defaults to the upper-left corner of the symbol. You can assign an origin or change the current origin with the Symbol Origin command in the Symbol Editor.

Saving Symbols

You can store a symbol in a library for use in many designs, or keep it in the design directory for use in a specific design.

If you're saving a new symbol, you're prompted for a name. If you're editing an existing symbol, changes are saved to the existing file.

You can use the Save As command to save the symbol with another file name, which is useful when you're designing several similar symbols. Save the original, modify it, then save the new version with a new name.

Symbol files have the extension `.sym`. The Symbol Editor adds this extension automatically when you specify the base name. If you specify a different extension, the Editor replaces it with `.sym`.

Printing Symbols

Use the Print command on the File menu to print a symbol. The default orientation is *landscape* (long side of the page horizontal). If *portrait* orientation (long side of the page vertical) would allow a larger image, choose the Page Setup command from the File menu to change the orientation.

Checking Symbols

You can check symbols for errors at any time. The Symbol Editor writes an error report to a file and displays it after the check is run. Clicking an error in the list highlights the error in the drawing.

The following types of errors are detected and reported:

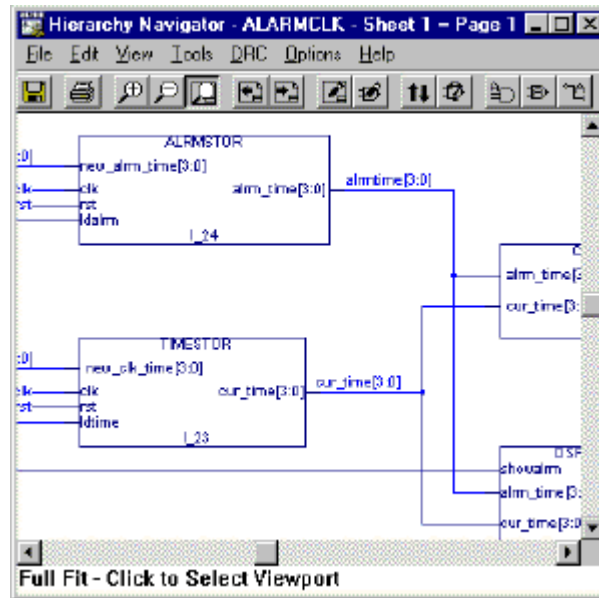
- Block symbols should have a schematic with the same name in the current directory.
- Symbols of type other than Block are usually primitives and should not have a schematic of the same name in the current directory.
- Each pin should have a PinName in a Block or Cell, and a PinNumber in a Gate or Component.
- Pins in Component and Pin symbols can only have one PinNumber.
- Pins in the same group of a Gate must all have the same Polarity, Load and Drive.
- Pins on Block symbols should not have Load or Drive specifications.
- Pins on non-Block symbols should have Load or Drive specified. Input pins should specify Load but not Drive. Output pins should specify Drive,

unless the pin is tristate. In that case, Load should also be specified, representing the load in the High-Z state. Bidirectional pins should specify both Load and Drive.

The Hierarchy Navigator

The Hierarchy Navigator program allows you to navigate through a schematic design consisting of a top-level schematic and lower-level schematics and HDL modules.

The Hierarchy Navigator loads a full hierarchical design all at once so that you can view it in its complete form, rather than as individual sources. Every schematic sheet and behavioral file at all levels of hierarchy is included. You can trace signals and connectivity throughout the design.



Attributes

An attribute is a characteristic or property belonging to, or associated with, a symbol, pin, or net. For example, attributes can describe:

- Number of connections to a block
- Delay from input to output
- Length of time taken to design a symbol

Attribute Modifiers

An *attribute modifier* specifies the conditions under which an attribute's value can be modified. The attribute modifiers are grouped based on where you can edit their values.

You can edit symbol and pin attribute values on the symbol and override the values in any schematic where the symbol appears. The four attribute modifiers described below control how attribute values can be changed in the schematic.

Modifier	Edit In	Description
<blank>	Anywhere in design	These attributes can be assigned or edited in the Symbol Editor, Schematic Editor, or Hierarchy Navigator.
!	Not Editable	Certain attributes are editable only by special "system" commands, such as Instance Names and Net Name Flags. In addition, the ini file may contain attributes that are not editable for the purposes of maintaining compatibility with other versions of the schematic (for example, for a different FPGA or ASIC device family) without losing the attribute name association.

These attributes are not listed in the Hierarchy Navigator attribute editor.

Modifier	Edit In	Description
-	Symbol Only	These attributes can only be assigned or modified in the symbol editor. They establish fixed values for all instances of the symbols to which they are attached. "symbol only" attributes will be listed in the symbol editor attribute editor, but not in the schematic editor or hierarchy navigator. This modifier cannot be assigned to net attributes.
\$	Symbol or Schematic	Attributes designated with this modifier can be assigned or modified in the Symbol Editor or Schematic Editor; they are not editable in the Hierarchy Navigator. These are typically used in conjunction with netlists that run from the schematic. Since those netlists do not have access to the hierarchical database, any attributes added through the Hierarchy Navigator would be lost.
*	Derived	Derived attributes can be assigned or modified anywhere in the design through the Symbol Editor, Schematic Editor, or Hierarchy Navigator.

Note: Attributes 00 through 99 are reserved definitions. Do not change their numbers or use. Attributes 100 through 199 are available for you to define and use for any purpose.

Attribute Window

Attribute values are displayed in *attribute windows*. Attribute values cannot be displayed unless the symbol has at least one attribute window.

You add attribute windows to a symbol when you define the symbol. Each window is assigned a unique number and the default attribute that will be displayed in that window. (The window number *does not* have to match the number of the assigned attribute.) When the symbol is placed in a schematic, the value of the assigned attribute appears in the window.

You can temporarily change which attribute is displayed in an attribute window, using the Attribute Display command from the Options menu. This is useful when you need to view attributes that are not currently displayed.

Attribute windows in schematics can be repositioned, one at a time, with the Attribute Location command from the Edit menu. Repositioning can make a crowded schematic more readable.

Note: *Attribute windows do not have visible outlines. Rather, they are predefined areas on or near the symbol.*

Attribute Functions

The principal source of information about a symbol's electrical characteristics and behavior is the attribute values attached to it. Your simulator uses these attributes to analyze and simulate the schematics you design.

Attribute values in a symbol definition become the default values for each symbol instance. These values are frequently overridden in the completed design, usually to optimize its performance.

Attributes that apply to all instances of a symbol (such as the vendor part number and the pin polarity) are generally assigned values when the symbol is created. Attributes that apply to a single instance (such as the instance name) are assigned after a symbol has been placed in the design.

Note: *The symbol libraries supplied with the ispLEVER software have predefined values for all the attributes required by the Verilog simulator.*

Attribute Types

There are four attribute types:

- | | |
|---------------|--|
| Global | Global attributes are constants such as feature size, supply voltage, or identification codes. These attributes are accessible from every sheet of every schematic at every level of hierarchy. |
| Symbol | Symbol attributes describe features related to the whole symbol. Examples are the width and length parameters of transistors, or SPICE-model characteristics. Symbol attributes usually apply only to the symbol on which they appear. |

Pin	Pin attributes describe features related to individual pins. Polarity, lead number, drive capability, and loading are typical pin attributes. Pin attributes are accessible at the instance level and can be modified in both the Schematic Editor and Hierarchy Navigator.
Net	Net attributes describe characteristics associated with nets. A good example is the stray capacitance of a net routed across a chip.

Attribute Names

An attribute's *name* identifies it to you. Attribute names are text strings and can contain any characters except spaces. Names are not case sensitive. You can mix cases to improve readability. Width, Length, RefDes, and PinNumber are examples of attribute names.

Attribute Numbers

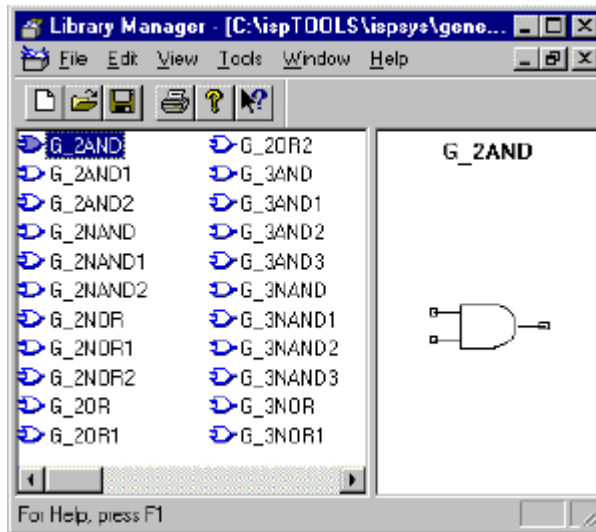
The attribute's *number* identifies it to the editors and the Hierarchy Navigator. The editors and Hierarchy Navigator use the number, *not* the name, to reference an attribute. This allows a different name to be assigned without changing the meaning or use of the attribute. The connection between an attribute's name and its number is defined in the initialization file.

Attribute Values

An attribute can be assigned a *value*. (Attributes 0-99 are reserved for the editors, the Hierarchy Navigator, and simulation. Most of them have predefined meanings.) A value is usually a number or a text string.

The Library Manager

The Library Manager lets you manage libraries of symbols that are used in your designs. The Library Manager also allows you to browse these libraries and to maintain them by adding, deleting, copying, and renaming the symbols in the libraries.



Why Use the Library Manager?

Using the Library Manager, you can clean up your folder structure by organizing your symbols into binary libraries, which use disk space more efficiently than separate symbol files.

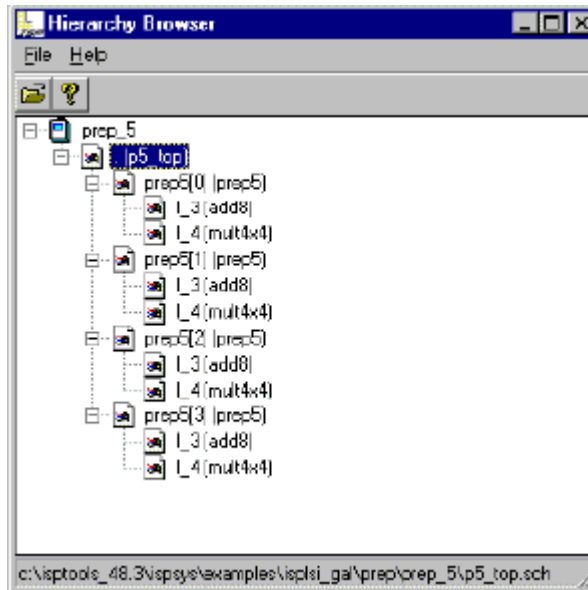
Working with Binary Symbol Libraries

There are two types of symbol libraries: *folder* and *binary*. Folder libraries are simply directories that contain symbols. Folder libraries and the symbols they contain can be manipulated using the Windows File Manager or Explorer.

A binary library is a symbol library that has been compressed into one compact file with the extension `.lib`, but it can contain many different symbols. *Using the Library Manager can create only binary libraries.*

The Hierarchy Browser

The Hierarchy Browser allows you to navigate through a design consisting of any combination of schematic and HDL modules. In contrast with the Hierarchy Navigator, the Hierarchy Browser works with designs whose top-level is either a schematic or HDL source. Additionally, you can cross probe between design sources and their appropriate tool.



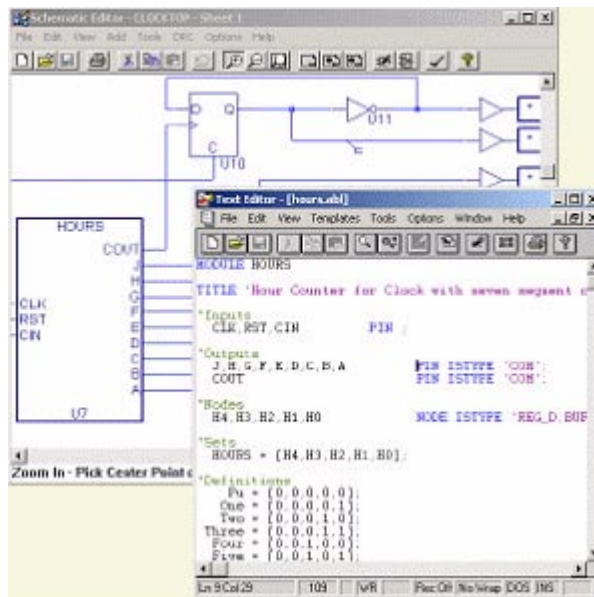
Introduction to the Hierarchy Browser

A large, complex design is typically broken into components, or *modules*. Circuitry for a specific function or interface can be abstracted as a separate module. The design represented by the module is said to be one level below the design in which the module appears. Or, the design is one level above the module.

Any design with more than one level is called a hierarchical design. The Hierarchy Browser shows all the design files (sources) associated with a project. There are several kinds of design sources in the ispLEVER software, including schematics, ABEL-HDL modules, VHDL modules, and Verilog HDL modules. The type of a source is indicated by the icon to the left of the instance name in the Hierarchy Browser.

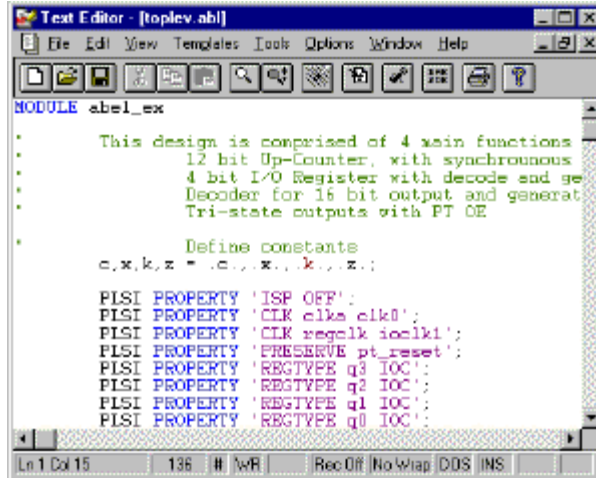
Mixed-Mode Design

The ispLEVER software supports mixed-mode design entry: a design with at least one schematic module as the top project source, and one or more sources of the same language. The language sources are mutually exclusive, so you must choose one of the three types when you begin a new project. For example, a schematic and an ABEL-HDL source, a Verilog HDL source, or a VHDL source.



The Text Editor

The Text Editor is the ispLEVER text entry tool. You use this tool to create and edit text-based files, such as ABEL-HDL files, test files, and project documentation files.



```
Text Editor - [toplev.abl]
File Edit View Templates Tools Options Window Help
This design is comprised of 4 main functions
.   12 bit Up-Counter, with synchronous
.   4 bit I/O Register with decode and ge
.   Decoder for 16 bit output and generat
.   Tri-state outputs with PT OE
.
.   Define constants
c,x,k,z = c,x,k,z;
PLSI PROPERTY 'ISP OFF';
PLSI PROPERTY 'CLK clk0 clk0';
PLSI PROPERTY 'CLK reqclk ioclk1';
PLSI PROPERTY 'PRESERVE pt_reset';
PLSI PROPERTY 'REGTYPE q3 IOC';
PLSI PROPERTY 'REGTYPE q2 IOC';
PLSI PROPERTY 'REGTYPE q1 IOC';
PLSI PROPERTY 'REGTYPE q0 IOC';
Ln 1 Col 15      136  #  NoWR  Rec Off No Wrap DDS INS
```

Design Simulation

Functional simulation is the process of simulating the functionality of your RTL design *before* synthesis, thus letting you find and correct basic design errors sooner. While functional simulation will verify your Boolean equations, it does not indicate timing problems.

The ispLEVER software supports functional simulation for any Lattice Semiconductor device using Lattice Logic Simulator or ModelSim™ from Model Technology. These simulators operate in both stand-alone and integrated environments.

Integrated Simulation

To simulate a design file inside the current project, the ispLEVER software provides integrated simulation. From the Project Navigator, you can run the appropriate process associated with the design file in the process window.

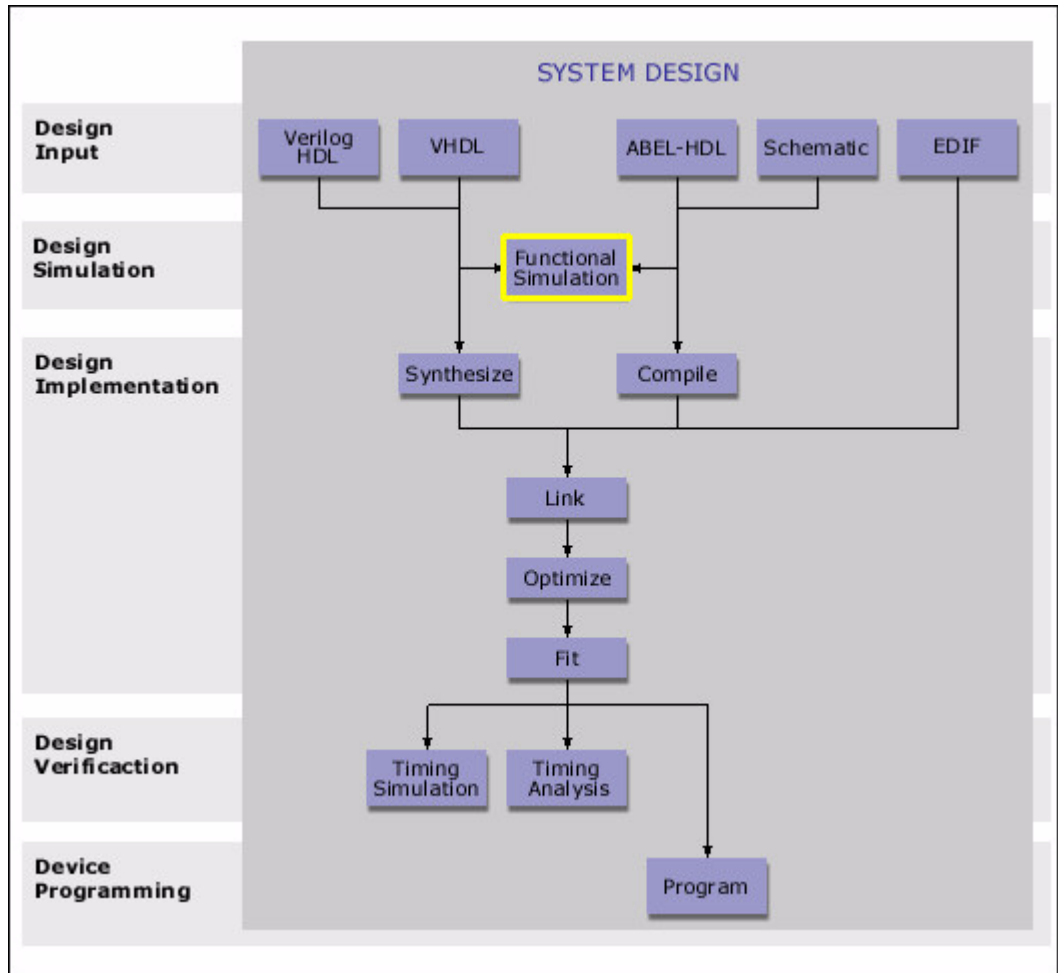
Project Navigator Process	Simulation Tool Used
Functional Simulation	Lattice Logic Simulator
Verilog Functional Simulation	ModelSim
VHDL Functional Simulation	ModelSim

Standalone Simulation

The ispLEVER software supports stand-alone functional simulation. This provides an easy entry if you need to simulate a design file outside the current project. Even if you have previously opened Lattice Logic Simulator or ModelSim from a project, you can change to the stand-alone mode flexibly by selecting the appropriate simulator from the Project Navigator **Tools** menu.

Functional Simulation - CPLD Process Flow

The figure below shows functional simulation within the CPLD process flow.

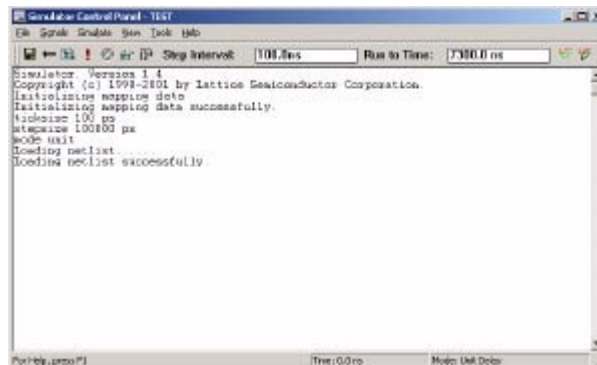


Lattice Logic Simulator

Lattice Logic Simulator performs logic simulation on your design and helps you verify the operation of your design before you implement it into a Lattice Semiconductor device. You can observe not only the gate-level behavior at its inputs and outputs, but also the behavior of internal nodes.

You can use Lattice Logic Simulator to verify a design:

- Before fitting (Functional Simulation)
- After fitting (Timing Simulation)



Simulation Support

The ispLEVER software supports simulation for Lattice ispLSI, ispMACH, ispGDX, and ispGDX2 device families. Lattice Logic Simulator operates in both standalone and integrated environments.

Stand-alone Simulation

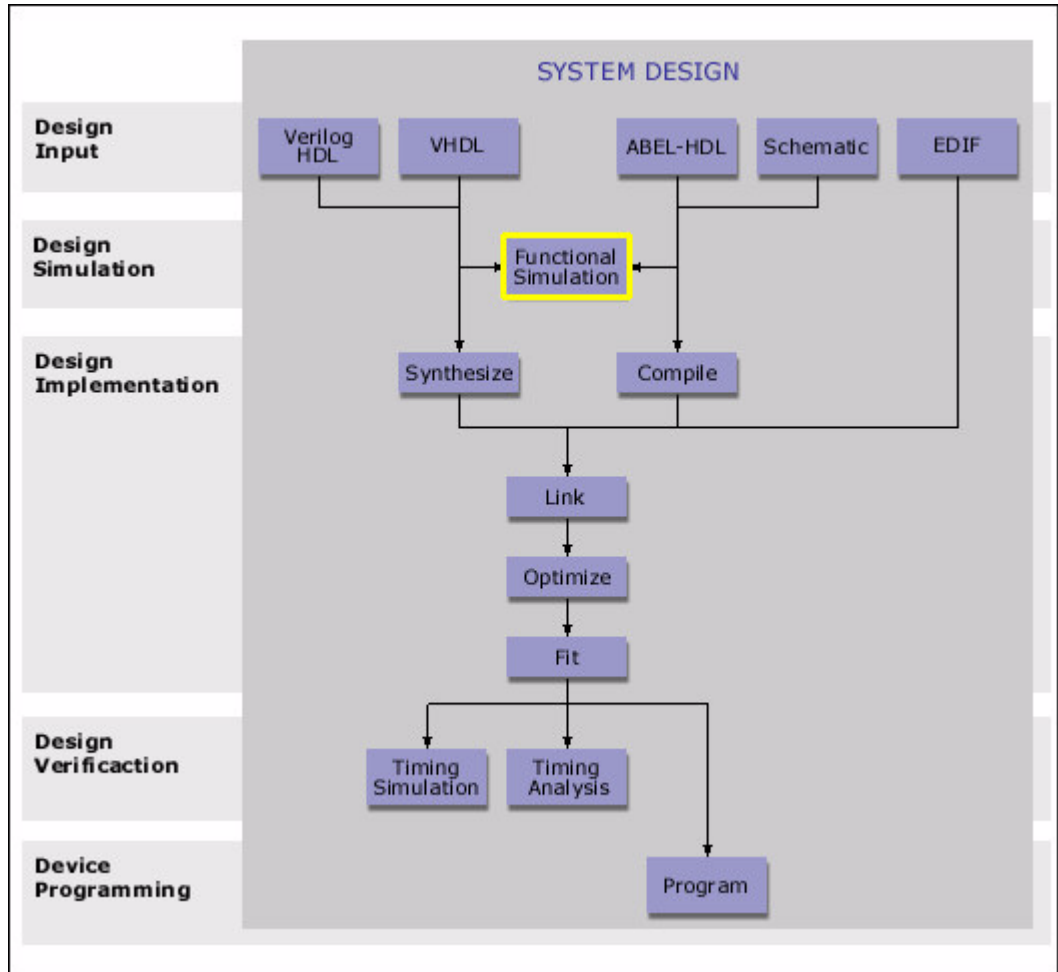
The ispLEVER software supports stand-alone functional and timing simulation. This is to provide an easy entry if you need to simulate a design file outside the current project. Even if you have previously opened Lattice Logic Simulator from a project, you can change to the stand-alone mode flexibly by choosing **Tools > Lattice Logic Simulator** from the Project Navigator.

Integrated Simulation

To simulate a design file inside the current project, the ispLEVER software provides integrated simulation. You can just run the Functional Simulation or Timing Simulation process associated with the design file in the Project Navigator Processes window.

Simulation in the ispLEVER Design Flow

The following figure displays Lattice Logic Simulator in the ispLEVER design flow.



Design Entry

Lattice Logic Simulator enables you to verify the operation of your design in the following formats:

- ABEL-HDL format (*design.abl*) - a hierarchical logic description language that supports a variety of behavioral input forms, including high-level equations, state diagrams, and truth tables.
- Schematic format (*design.sch*) - describes your circuit in terms of the components used and how they connect to each other.
- VHDL format (*design.vhd*) - Very High Speed IC Hardware Description Language format.

- Verilog HDL format (*design.v*) - an industry-standard hardware description language used to describe the behavior of hardware that can be implemented directly by logic synthesis tools.
- EDIF format (*design.edi*) – an industry-standard netlist file generated by synthesis tools.

The ispLEVER software also supports mixed design entry as follows:

- Mixed ABEL-HDL and Schematic entry
- Mixed Schematic and VHDL entry
- Mixed Schematic and Verilog HDL entry

Test Stimulus

Once you have completed your design (or a module of the design), you can test it to confirm that it behaves the way you expect it to. Simulation requires a test stimulus file that specifies the input waveforms.

Lattice Logic Simulator can accept two types of test stimulus files:

- Graphic waveform file
- Test vector file

The ispLEVER software has an interface with Model Technology's VHDL and Verilog Simulator that accepts the following two stimulus files:

- VHDL test bench file
- Verilog test fixture file

Install the ModelSim before you do the VHDL and Verilog HDL simulation. Refer to the ModelSim online help for more information on creating VHDL test bench and Verilog test fixture.

Creating Test Stimulus for Lattice Logic Simulator

Before simulation, you must create a stimulus file that specifies the input waveforms.

- Creating a graphic waveform file

The Waveform Editor lets you graphically create input stimulus waveforms for your design by drawing them directly on the screen. The stimuli can be edited graphically or by modifying values in dialog boxes. The Waveform Editor then converts the waveforms into a stimulus file that the simulator recognizes. Waveform files (in Waveform Description Language format) are also useful as input to automatic test equipment or as documentation of the circuit's expected behavior.

If you associate the waveform stimulus file (*.wdl*) with the selected device in your design, both the functional and timing simulation processes are supported. However, if you associate the waveform stimulus file with a design module, only functional simulation is available.

- Creating a test vector file

You can create a test vector file in a text editor using proper keywords. Test vectors are sets of input stimulus values and corresponding expected outputs that can be used with both functional and timing simulators. Test vectors can be specified either in a top-level ABEL-HDL source or in a separate ABEL-HDL test vector format file called a `.abv` file. The `.abv` file is considered a text document and is kept above the device level in the Sources window. Whether the test vectors are part of a top-level ABEL-HDL source (`.abl`) or are in a separate file, they will be compiled and passed to the simulator.

Creating Graphic Waveforms

You can graphically create a waveform file using the Waveform Editor, which can be used as a stand-alone tool or interactively with the Hierarchy Navigator. Running the Waveform Editor with the Hierarchy Navigator offers a number of advantages, including direct entry of node names and the direct display of stimulus logic levels on the schematic.

The Waveform Editor uses a data model called the Waveform Description Language (WDL). The language represents a waveform as a sequence of signal states separated by time intervals. The language also has constructs that let you express the waveform pattern hierarchically. You do not have to be familiar with the Waveform Description Language to use the Waveform Editor.

Creating Test Vectors

Test vectors can be specified in two ways:

- Specified in an ABEL-HDL source (`.abl`)

The most common method is to place test vectors in the ABEL-HDL source file. If you use this method, the Project Navigator will detect the presence of test vectors in the source file and create a "dummy" test vector file. This file indicates to the system that the actual test vectors are in the ABEL-HDL source file.

- Specified in an external Test Vector file (`.abv`).

Placing test vectors in an ABV file instead of in an ABEL-HDL source file improves processing time. By placing test vectors in an ABV file you will be able to change the test vectors and re-simulate without having to recompile the logic.

Creating a VHDL Test Bench

Before you do VHDL simulation, you need to create a test bench file as the test stimulus that specifies the input for simulation.

You can manually create a VHDL Test Bench File (`*.vhd`), use a VHDL Test Bench Template File (`*.vht`), or export a VHDL Test Bench File with the Waveform Editor.

Creating a Verilog Test Bench

Before you do Verilog HDL simulation, you need to create a test fixture file as the test stimulus that specifies the input for simulation.

You can either manually create a Verilog Test Fixture File (* .t_f) or include a Verilog Test Fixture Template (* .t_fi) into the test fixture.

Special Constants

- .C. Translates to 0, 1, 0 and time unit delay is 3
- .K. Translates to 1, 0, 1 and time unit delay is 3
- .D. Translates to 1,0 and time unit delay is 2
- .U. Translates to 0, 1 and time unit delay is 2
- .F. Translates to .Z.
- .SV#. #2-9 translates to 0 (can be changed in .ini file)
- .P. Assigns the value provided in vectors to signals. When .C. .K. .U. .D. appear as well as .P., they are translated to '0'

The .P. in .abv means the supervoltage preload. It is used to load registers to the desired state. If the registers do not set preload value when CLK is set to .P., the simulation result is complex and unpredictable.

.P. takes effect only when it is used as .clk of a flip-flop. If it is not used as .clk, it is translated to '0', and the simulator does not care where the .P. appears; it is only concerned whether the .P. appears, which means the simulator assumes that you used the .P. correctly.

- .R. Specifies random test vector values in test vector blocks. Translates to 0 or 1 randomly.

Note: All the special constants, except .R., cannot be used together with the CYCLE or WAIT keyword.

Using .P. to Pre-load Values in the State Machine

.P. is a special constant used to pre-load values to registers. The main function of .P. in the test stimulus is to preload some invalid values to test the function of a design, especially for some invalid state of the state machine.

Note: .P. cannot be used together with the CYCLE or WAIT keyword.

The following example shows the usage of .P.:

```
MODULE preload;
TITLE 'Using Special Constant .P. to show state transition'
```

```
"Inputs
clk    pin;
reset  pin;

"Outputs
[q0..q2]  pin  istype 'reg';

"State Register
sreg = [q2..q0];

"State
A = [0, 0, 0];
B = [0, 0, 1];
C = [0, 1, 0];
D = [0, 1, 1];
E = [1, 0, 0];

"Illegal states
F = [1, 0, 1];
G = [1, 1, 0];
H = [1, 1, 1];

Equations
sreg.clk = clk;
sreg.re = reset;

State_diagram sreg;
State A: goto B;
State B: goto C;
State C: goto D;
State D: goto E;
State E: goto A;

"Illegal state
State F: goto A;
State G: goto A;
```

```

State H: goto A;

"Nomal
Test_vectors ([reset, clk] -> sreg);
    [1, 0 ] -> 0;
    @repeat 8 {
    [0, .c.] -> .x.;};

"Preload to test invalid state
Test_vectors([reset, clk, [sreg]] -> sreg)
    [1, 0, [.x.]] -> 0;
    [0, .P., [ F ]] -> F;
    [0, 0, [.x.]] -> .x.;
    [0, .c., [.x.]] -> A;

    [0, .P., [ G ]] -> G;
    [0, 0, [.x.]] -> .x.;
    [0, .c., [.x.]] -> A;

    [0, .P., [ H ]] -> H;
    [0, 0, [.x.]] -> .x.;
    [0, .c., [.x.]] -> A;

End;

```

Viewing Simulation Results

The Waveform Viewer is the primary tool for viewing simulation results. The Viewer graphically depicts the activity on any node in the simulation database. The Viewer is automatically updated as simulation progresses. Several tools are provided to analyze the results.

The Waveform Viewer also works in conjunction with the Hierarchy Navigator to display simulation results directly on the schematic. This feature is called "cross probing."

Cross Probing

Cross probing lets you view simulation results on schematics and lets you add waveforms to the Waveform Viewer from a schematic. Crossing probing makes it easier to correlate simulation results with the sources in the design.

Note: *Functional simulation reduces your ability to cross probe. Cross probing relies on direct correspondence between the node names in sources and those*

in the simulation database. Synthesis and optimization often eliminate signals, change names, and remove pieces of logic. These changes interfere with the resolution of name references. You can still cross-probe device I/O pins, since these are never removed. This is not the case with many internal nodes. You have to browse the simulation database directly within the Waveform Viewer to select internal nodes.

Simulation Mode

The simulation mode determines how the simulator handles delays in the simulation netlists. The option modes are Inertial and Transport.

Showing Logic Values on Schematics

The following table lists the states displayed on the schematic and the logic values in Lattice Logic Simulator they correspond to:

<u>Schematic Value</u>	<u>Logic Value</u>	<u>Strength Value</u>
0	0	supply0, strong0, pull0, weak0
1	1	supply1, strong1, pull1, weak1
Z	Z	all strengths
X	X	all strengths

Using Query to Navigate

The Query command in the Hierarchy Navigator is another useful debugging tool. The Query command lets you interrogate the Navigator's online connectivity database by clicking items in the schematic. Highlighting an item in the Viewer displays the associated schematic and positions the cursor over that item.

For example, querying a net generates a list of all the instances and pins in the hierarchy that connect to that net. Selecting an instance from the list displays the schematic with that instance. With the instance highlighted, you can trace a signal throughout the entire design by selecting items from the list.

Design Implementation

Synthesizing and Compiling

The ispLEVER software accepts several design entry formats. With the exception of EDIF, all designs must be either synthesized or compiled before going to the Fitter.

For Verilog and VHDL designs, the ispLEVER software provides two synthesis tools that are integrated into the Project Navigator environment: Synplicity Synplify and Exemplar LeonardoSpectrum. You can synthesize your Verilog or VHDL design as a standalone process by choosing the synthesis tool from the Lattice Semiconductor program group in your Start menu, or you can synthesize automatically and seamlessly within the Project Navigator.

For ABEL-HDL and Schematic designs, the compilation process is an integrated part of the ispLEVER process flow. When you compile a design, you are changing your design entry format into Boolean equations, which serve as input to simulation and device implementation programs. In general, compiling a design involves running every process after design entry. These processes include compiling and optimizing steps that can be performed on a single source or on the entire design.

Keeping Track of Processes

The Project Navigator automatically keeps track of your design's processes for you. For example, it knows which processes should be run for a targeted device, a selected source, or for the entire design. Also, you can choose to run any process step and the Project Navigator will run all other processes required to complete that selected step, but not run further, unnecessary steps.

The Project Navigator lists all processes for a selected source in the Processes window. Device-related processes, such as fitting the design, are shown in the Processes window after you select a target device, and highlight it in the Sources window.

Understanding the Compilation Process

The Project Navigator processes each logic module, schematic file, or EDIF netlist to obtain an intermediate file that can later be linked together before fitting the design into a Lattice device.

There are more processes required to compile a logic source than a schematic source, primarily because logic designs are language-based and are stored in ASCII format. This means that the ispLEVER software must check the language syntax and process equation statements that are within the logic file.

The processing steps required to compile a design are listed below in the order in which they run:

- Compile (for logic, schematic, EDIF, or test vector files)
- Check Syntax (for logic files)
- Compiler Listing (for logic files)
- Compiled Equations (for logic, schematic, and EDIF files)
- Signal Cross Reference (for EDIF files)

Compile (Logic or Schematic)

This comprehensive process compiles a logic module, a schematic design file, or an EDIF netlist. Design compilation steps differ between source types, as described below:

Compile Logic (for logic sources)

- Checks for and flags syntax errors
- Converts state diagrams and truth tables into equations
- Expands macros
- Converts equations with sets to equations without sets
- Replaces all operators with equivalent operations using only NOTs, ANDs, ORs and XORs
- ORs together equations that cause multiple assignments to the same identifier
- Performs simple logic reduction
- Translates the equations into the OPEN-ABEL-2.0 file format

Compile Schematic (for schematic sources)

- Compiles the schematic to produce a BLIF format file, including any attributes or properties specified. Schematics compiled this way should use only the Device-Independent symbol library.

Note: You should run design rule checking (Schematic Editor: **DRC > Consistency Check**) before compiling the schematic.

Compile EDIF (EDIF)

- Compiles the EDIF file to produce a BLIF format file, including any attributes or properties specified.

Check Syntax

Checks the syntax of a logic module. No compilation is run. If there are syntax errors, the errors can be viewed in the Process Log File. If you want to see the errors in a compiler listing report format, use the Compiler Listing process.

Compiler Listing

The Compiler Listing process gives a record of the compilation of your source file. The report shows your logic file by line numbers, with errors and warnings below the line on which they occurred.

Compiled Equations

This process shows the Boolean equations produced by the compiler. The equations are shown in sum-of-products form. Positive and reverse polarity equations are displayed, along with product term and fan-in/fan-out summaries for each signal.

Signal Cross-Reference (EDIF)

This option displays a cross-reference of the old names to the new names (converting long and hierarchical to mangled names). Names that are more than 32 characters long or that contain one or more of the characters '/', '>', or '@' will not be displayed properly.

Process Options

For many processes, there are processing options that you can specify. These options include compiler options, such as custom arguments or processing changes, and optimization options, such as node collapsing.

You use *properties* to specify these options. The properties available at any given time depend on the selected source file, the selected process, and the targeted device.

Viewing and Setting Process Option Properties

You can view current properties or set new properties for a process.

Compiling Source Files

The Project Navigator Auto-Update feature reprocesses sources when they are needed to perform the process you request.

However, you can compile individual source files by selecting the file in the Sources window, and then double-clicking **Compile Logic** in the Processes window. Alternatively, you can double-click a report in the Processes window, and the software will compile the source automatically.

Optimizing a Design

The default options in the ispLEVER software are set up to achieve the highest possible performance in the smallest possible device, for most designs. You can choose to maximize design flexibility by spreading out logic or exercise tighter control over the fitting process to achieve your design goals.

Each clock signal is evaluated and classified as a global clock or a non-global clock. The Fitter attempts to place all global clock signals at global clock pins (check the log file for the status of all clock signals after optimization). The Fitter assigns all other clock signals to I/O pins and implements them as Product Term clocks, if the architecture supports Product Term clocks. Input pins and nodes that are defined but not referenced (not used by another equation) are discarded from the design during optimization (warning messages are generated).

Design Resources Check

Information about the internal architecture of the specified device is loaded and resource checks are performed on the design. Errors are reported if the design exceeds the device's product term, macrocell, pin, clock, set, reset, or output enable control resources.

Logic Synthesis Options

Logic Synthesis options allow you to control how logic functions are optimized before partitioning takes place.

Boolean Logic Reduction

This option removes redundant product terms from each equation. Unless your equations have redundant logic to prevent problems (for example, in combinatorial functions), you should always leave this option selected.

D/T Synthesis

This option lets the optimizer automatically choose between a D-Type or T-Type register, thereby reducing the product term requirements. In some cases, the speed of the design may improve if only D-Type registers are used in an M4 device. This option should be selected for most designs.

Input Register Optimization

This option allows the Fitter to automatically place single-variable registered functions in input pad registers. This option should be selected for M4 devices unless you are trying to prevent the use of input registers.

XOR Synthesis

This option enables or disables exclusive OR synthesis. When this option is selected, the optimizer synthesizes XOR equations, if this can be achieved in the design. When this option is cleared, the sum-of-product equations will be generated. This option is device-dependent. Default state = Enabled.

Node Collapsing

This option allows the optimizer to collapse intermediate combinatorial nodes into registers and output pins, thus speeding up the design. Unless you have handcrafted each equation in your design, you should leave this option selected. This option should always be selected for designs that have been synthesized or described in low-level combinatorial gates.

Speed

This option collapses all nodes up to the set Product Term limit, globally optimized, without regard for the path.

Area

This option collapses all nodes up to the set Product Term limit, without increasing area cost.

Fmax

This option causes the Logic Optimizer to automatically identify all critical paths between any pair of registers, from clock-pin of one register to data-pin of the other register (or the same register). The Logic Optimizer then attempts to collapse/combine the logic nodes along the critical paths, reduce the logic level, and allow the chip to run at a higher frequency.

Collapsing Max. Product Term

This option lets you control the Fitter optimization process by setting a maximum limit on the number of Product Terms (PT) in each equation. In other words, the Optimizer shapes the equations relative to the set number of PT. For example, if the value is set to 35, the Optimizer stops collapsing equations when it exceeds 35 PT.

This option works the opposite of Splitting Max. Product Term.

Collapsing Max. Input

This option lets you control the Fitter optimization process by setting a maximum limit on the number of inputs in each equation. For example, if the value is set to 32, the Optimizer stops collapsing inputs when it exceeds 32 inputs.

Splitting Max. Product Term

This option lets you control the Fitter optimization process by setting a maximum limit on the number of Product Terms (PT) in each equation. In other words, the Optimizer shapes the equations relative to the set number of PT. For example, if the value is set to 35, the Optimizer splits equations if it has more than 35 PT.

This option works the opposite of Collapsing Max. Product Term.

Example

An M4-32 design consists of six equations having 12 product terms each, and one equation having 21 product terms. (An M4-32 macrocell can implement up to 20 product terms without equation splitting.) The Fitter can implement each

of the six smaller equations as single-macrocell equations, but the one larger equation must be implemented using two macrocells. In its default mode, the optimizer will split the 21-product term equation into one equation of 20 product terms and one equation of 2 product terms (the extra product term is required to accept feedback from the second macrocell).

Reducing the equation-splitting threshold to 12 will result in less of an imbalance in the number of product terms placed at each macrocell. Each of the original 12 product term equations remains at a single macrocell, while the 21 product term equations is split into two macrocells: one with 12 product terms and one with 10 product terms. Thus, none of the equations are using the maximum capacity of its macrocell, which improves the odds of fitting the design and makes it easier to add logic to the design later.

Note: Do not reduce the equation splitting threshold if doing so will cause many equations to be split. If, for instance, the preceding example's six smaller equations had contained 15 product terms each, setting the gate-splitting threshold to 12 would have caused all seven equations to be split, resulting in 16 under-utilized macrocells.

Example

Consider the following:

- A synchronous registered equation with 22 product terms
- Splitting Max. Product Term field set to 20

The equation will be split into two equations, one with 20 product terms and one with 3 product terms. It will take two passes through the array to implement the new equations.

Setting Logic Synthesis Options

You can set logic synthesis options using the Logic Synthesis tab on the Global Constraints dialog box.

Utilization Options

Utilization options let you specify the percentage of device resources available during each fitter run. You can choose to reduce the device resources available during the initial fitter run, back annotate the pins, and then increase the available device resources when making design changes.

Reducing available device resources during the fitter run may increase the fitter runtime. For example, when the maximum number of block inputs is set too low (<60%). This condition may cause the Fitter to take a long time grouping (i.e. partitioning) logic equations into blocks because the blocks have fewer available resources.

Logic Grouping

You can use logic grouping to exercise manual control over the partitioner. Logic grouping lets you manually pack selected portions of your design into the same block while setting up the global optimization options to spread out the rest of the logic.

Logic grouping can also be used to group selected inputs, outputs, and buried logic functions into the same block or segment to achieve performance goals.

In cases where the partitioner is unable to find a solution, manually grouping a small portion of the design may aid in the fitting process.

If you do attempt manual grouping, try to place logic with common inputs and feedback in the same block. This minimizes the number of signals crossing between blocks, which results in a lower demand for interconnection resources and an increased likelihood of a successful fit.

Fitting a Design

The ispLEVER software has a single user interface with all options preset to deliver the highest possible push-button performance. At the end of a successful fitter run, the ispLEVER software generates a JEDEC file, as well as a fitter report, so that you can see how the ispLEVER software has routed the design and utilized resources on the part.

Assigning Pin and Node Locations

The ispLEVER software lets you pre-assign pin and node locations. You can use the Location Assignment dialog box in the Constraint Editor to assign input and output pins and buried nodes. The Macrocell, Block, and Segment list boxes are context sensitive to the selected device; only applicable features are available.

You can also use the drag and drop feature in the Package View of the Constraint Editor to assign input, output and bidirectional pins.

Pin and Node Pre-Assignment

Pre-assigning pins lets you lay out your board at the same time as you are doing logic design, thus shortening the design cycle. Pre-assigning nodes is usually not required and is not recommended.

Pin Assignment Guidelines

If you want to pre-place signals (not recommended unless pinout configuration is important), follow these guidelines:

- Do not place large equations to macrocells or pins at the beginning or end of a block.
- Signals that share many common inputs should generally be grouped in the same block (the Partitioner does this automatically). Signals that do not share many common inputs should generally be distributed across several blocks to avoid overburdening the switch matrix for a single block.

Large Functions at the End of a Block

The macrocells at the end of a block have access to fewer product terms than other macrocells.

- Cell number 0, the first cell in all devices, can access the product term clusters from adjacent, higher-numbered cells, but it cannot access any lower-numbered cells (cell 0 being the lowest-numbered cell in the block).
- The last cell in a block can access the product term cluster from the adjacent lower-numbered cell, but it cannot access any higher-numbered cells.

If signals have not been assigned to macrocells, the Fitter will find a macrocell replacement for all the signals that satisfy their product term requirements.

Adjacent Macrocell Use

In MACH devices, adjacent macrocells can share clusters. Therefore, with designs having equations that use a high number of product terms, it is a good idea not to place them in adjacent macrocells.

Modifying Assignments

You may want to modify current pin or group assignments.

Deleting Assignments

You can delete project assignments via the Constraint Editor. To do this, select the entire row whose existing assignment(s) you want to delete. From the Edit menu, select **Delete Row(s)**. You may also right-click and select **Delete Row(s)**. In cases where you no longer want any of the current assignments, you can delete all of them at the same time.

Ignoring Assignments

There may be times when you want to ignore, but not delete, the current assignments. For example, after you complete a design, you may want to try fitting it into a different device. In this case, the current pin assignments may not be valid for the new device. The ispLEVER software lets you ignore current constraints for the next Fitter run.

Power Control

Using the ispLEVER software, you can control power settings for your device. By default, the device is always set to high power, high speed. However, you can set the device or blocks of the device to low power mode. This setting results in slightly decreased speed, but increased power savings. This is useful for handheld and battery-operated devices.

Slew Rate Control

For the majority of Lattice devices, you can set the slew rate to either Slow or Fast. By default, the slew rate is set to Fast. However, changing it to Slow can result in less board noise.

Partitioning

After optimization, the design is partitioned into individual blocks on the specified device. Partitioning is achieved by assigning logic to specific blocks, based on the following considerations:

- Individual signal pre-placements and Grouping assignments
- A block's available internal resources (free macrocells, product terms, clock signals, and so forth)
- The switch-matrix interconnect resources available to the block

The Partitioner considers commonality of signals, macrocell requirements, Set/Reset requirements, product-term requirements, and other factors to determine which partition is most likely to succeed in fitting the design. Only

partitions that are likely to succeed (according to the Partitioner's rules) are attempted.

Balanced Partitioning

Controlling how the Partitioner works can be very important. There is one important strategy for partitioning and that is called Balanced Partitioning. By selecting the Balanced Partitioning option in the Global Constraints dialog box, you are telling the Partitioner to spread all of the signals among all the blocks in the device, rather than trying to fill a few blocks to their maximum potential.

There are advantages to either side of the strategy. If you turn balanced partitioning on, you can save room in the device for any future functionality you might want to add to existing logic. However, turning balanced partitioning off lets you "pack" as much logic into the minimum number of blocks in the device as possible, leaving some free blocks for future design enhancements.

Fitting (Place and Route)

Placement is the assignment of physical block resources such as I/O pins, XORs, registers, and product-term clusters to logic equations. *Routing* is the assignment of switch-matrix interconnect resources to logic equations, after the logic equations are placed.

Placement

In the placement phase of the fitting process, individual equations are assigned to physical resources, as follows:

- Logic equations that have been pre-assigned to pins are assigned first.
- Buried logic functions are placed in the remaining unused macrocells.
- Inputs are assigned to any available pin. These pins can be dedicated inputs pins, clock/input pins, or I/O pins that correspond to macrocells that are either unused or used to implement buried logic functions.
- Outputs can be assigned to any unused I/O pin.

Spread Placement

Controlling how the Placer works is also important. When you select the Spread Placement option, you are telling the Placer to spread the signals in the block as far out as possible.

Routing

In the routing phase, the Fitter attempts to route input, output, and feedback signals to and from the physical resources assigned in the placement phase. If the Fitter fails to route all signals, it tries another placement. The Fitter continues trying different placements, and different routing attempts within each placement, until a successful fit is found or the time allotted for fitting is exceeded.

Fitter Options

The Global Constraints dialog box lets you set options for the Fitter. Using the Global Constraints dialog box, you can tell the Fitter to pack as much logic into

the device as possible, spread the logic across the entire device, or use other advanced options such as specifying device utilization. The following sections describe these options.

Pack Design

The Pack Design option lets you pack as much logic into the device as possible. This option allows you to achieve the highest possible performance in the smallest possible device, for most designs. Each block may be completely filled, leaving less room for any design changes or logic additions.

Spread Design

The Spread Design option spreads all of the logic across the entire device rather than trying to fill each block to its maximum potential. This option allows you to achieve the highest possible performance, while leaving room for any additional functionality that you may want to add in the future. The fitter leaves room to accommodate design changes to existing logic. Because each block may be incompletely filled, the design may *or may not* require a larger device to achieve a successful fit.

Advanced Options

The advanced options let you individually control the partitioning and placement algorithms.

Balance Partitioning

The Balance Partitioning advanced option partitions the design evenly among all the blocks in the device, so each block should have the same amount of resources used. When this option is cleared, the software partitions the design block-by-block, filling up one block at a time. This means that some blocks may be filled up completely, while others may be unused.

Spread Placement

The Spread Placement advanced option places the signals evenly, or spreads them out, among macrocells in the block. Spreading out the placement lets you make minor changes to the existing output and node signals in the block. When this option is cleared, the software assigns design signals to the first available macrocell, making it easier to add new outputs or nodes to a block.

Fitter Effort

The Fitter Effort option is used to instruct the Fitter how much effort to apply to a fit. The Low option enables a faster fitting process, but will be more likely to result in failures to fit when the utilization gets higher. The High option provides the most exhaustive search of the solution space, but takes more time.

Fitter Report Formats

Two Fitter Report formats are available in the ispLEVER software, text and HTML.

- If you select the Fitter Report process associated with the target device, the Fitter Report is opened in the Output Panel of the Project Navigator or in the Report Viewer.

Note: By default, the ispLEVER software opens Fitter Report in the Output Panel of the Project Navigator. If you want it opened in the Report Viewer, select **Using Report Viewer** in the Log tab of the Environment Options dialog box (Project Navigator: **Options > Environment**).

- If you select the HTML Fitter Report process associated with the target device, the Fitter Report is opened with your local Internet Browser.

Formatting the Fitter Report

You can select various options that determine the information in the Fitter report using the Fitter Report Options dialog box.

The Fitter Report

The Fitter Report displays statistics and information on the fitting process of your design, including utilization numbers, pin assignments, etc. The Fitter Report is also written into HTML format to allow user to browse through the report easily.

The Fitter Report is divided into several sections, each briefly described below.

Project Summary

As the name implies, this section summarizes the design. It reports the name and location of the project, and the date it was fitted. This section also reports the targeted device and package, as well as the design source format.

Compilation Times

This section tells you how long it took the Fitter to fit the design in the specified device. The name for each process step is listed, as well as the total elapsed time. Prefit Time consists mainly of run-times of the design compilation and optimization phases. Total Fit Time is the total run-time of the design compilation, optimization, partition, placement and routing phases.

Design Summary

This section reports statistical information about the design, such as the number of Inputs, Outputs, Bidir Signals, Flip-flops, Registered Functions, Product Terms and Reserved Pins. It also points out the number of unique control signals in the design.

Device Resource Summary

This section lists all of the resources available within the device and how much of each resource has been used by the design. It also reports how much of each resource is still available.

GLB Resource Summary

This section lists various GLB (and segment) level resource counts, such as fanin (or array inputs), I/O pins, input registers, macrocells, logic product terms and product term clusters.

GLB Control Summary

This section lists the totals for all control signals, and how much of each is utilized by individual GLBs.

Optimizer and Fitter Options

This section displays all of the settings that were used to fit and optimize the design. These include things such as Ignoring Constraints to the type of flip-flop synthesis you have chosen. The information in this section is set with the Constraints Options dialog box.

Pinout Listing

This section lists the I/Os and control signals on the device, and how they are assigned.

(Input, Output, Bidir, Buried) Signal List

This section reports information on individual I/Os, such as I/O type, location assignment, the fan-out, and other signal attributes.

Signals Fan-out List

This section lists signal resources and the functions they fan-out to.

GLB (GLB name) Cluster Steering Tables

This section shows information about how functions and inputs are placed in a GLB. It shows how product terms are steered to a macrocell on which a function has been placed. It also contains information about what type of control signal has been used.

GLB (GLB name) Logic Array Fanin

This section shows how design signals are mapped to individual GLB block inputs.

Product Term Histogram

This section lists and sorts the equations according to the number of product terms they use (in the logic only).

GLB Input Histogram

This section lists and sorts the equations according to their number of inputs, which includes the logic and the ctrl signals, but not the global signals (dedicated routing).

Post-Fit Equations

This section reports the equations in your design, after fitting. It begins with a product term histogram and a GLB input histogram.

Backannotating Assignments

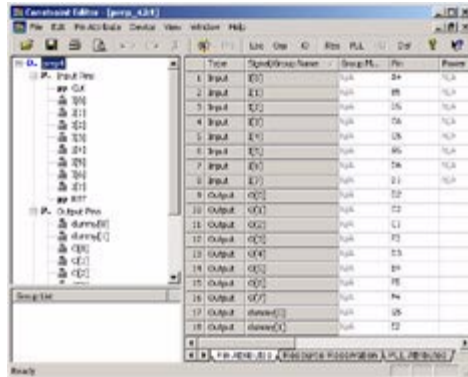
You can backannotate assignments from the Fitter output using the Backannotation tab on the Constraints Options dialog box. This feature lets you retain the assignments made by the Fitter so that they can be used in a future fitting process.

You can only backannotate project assignments after the "Fit Design" process has been successfully completed. An error message appears if the ispLEVER software detects that this process did not complete successfully.

The Constraint Editor

The Constraint Editor lets you specify pin and node assignments, group assignments, pin reservations, power level settings, and output slew-rates. It reads the constraint file and displays the constraint settings. Modifications to the constraint file are made via the function dialog boxes. Most of the attributes can be modified directly in the sheet.

Pin assignments can be set in the Package View with drag-and-drop functionality.



The Constraint Editor implements simple error checking to ensure that the user assignments or constraints are applicable to the selected device and that there are no conflicting assignments. If the user constraints do not apply to the selected device, or are conflicting with the selected device, the Constraint Editor displays these constraints in red. For instance, if the user changes the device type after specifying some pin assignments, the Constraint Editor will display the non-applicable pin assignments in red. The user can delete these constraints in the function dialog boxes or via **Tools > Clear Constraints** command in the Project Navigator.

Note that the design signal labels are only listed exclusively in the Location and Group Assignment dialog boxes. For instance, if Signal "ld_new_alarm_time" is assigned to a pin via the Location Assignment dialog box, then this signal will not be listed in the Group Assignment dialog box, and vice versa. This feature prevents conflicting assignments from being implemented.

Assigning Pins and Nodes

Pre-assigning pins and/or nodes allows designs to be fit with resources assigned to the same physical locations. You can add pre-assignments of pins and nodes to your design source files or make pin/node assignments in the Constraint Editor.

Assigning Signals to Groups

It is sometimes advantageous to group signals that have a logic association into the same logic block, such as segment and GLB. Grouping allows the Fitter to control where the signals are placed inside the logic block. Group assignments can be made in your design source files. You can also group signals using the Constraint Editor.

Node Preserving

When a design is processed in the ispLEVER software, the Pre-fitters will perform optimization on each of the sources, as well as the linked design, to try to minimize the logic needed. Logic can be manually partitioned in order to achieve speed and/or area requirements. There are a few methods that ensure that nodes are not removed or collapsed into others when the Optimizers are run. We call these methods *Node Preserving*. Node preserving has the opposite effects of *Node Collapsing*.

Resource Reservations

Sometimes it is necessary to reserve pins, GLBs or segments for future use. The ispLEVER software allows the reservation of logic resources in the devices. This feature ensures that the Optimizers do not use those reserved resources and that the Fitter does not use it when fitting the rest of the resources. You can define resource reservations in your design source files to reserve pins, GLBs or segments. The Constraint Editor also provides a feature for you to do resource reservation in its Resource Reservation dialog box.

Slew Rate

For the majority of Lattice devices, you can set the slew rate to either Slow or Fast. One advantage to changing slew rate to Slow is a reduction in board noise.

You can specify the slew rate of output and bidirectional pins in your design source files. Slew rate can also be set in the IO Types Setting dialog box of the Constraint Editor.

PULL

The I/O pins of the Lattice devices include internal circuitry to allow pin functions. These functions include pull-up resistors, pull-down resistors, and Bus-Friendly™ configurations. Using the "PULL" attribute can access these features. You can set the PULL attribute either in your design source files or in the IO Types Setting dialog box of the Constraint Editor.

Open Drain

The output and bi-directional pins of some Lattice devices can be set to an "Open Drain" configuration. As with most opendrain configurations, the pin drives low when it is logic 0 and is high-impedance when it is logic 1. You can assign opendrain configuration to pins in your design source files.

I/O Type

Some of the Lattice devices include the I/O Type feature. This feature allows the I/O pins to be configured to different I/O standards. You can specify I/O Types either in your design source files or in the Constraint Editor.

PLL

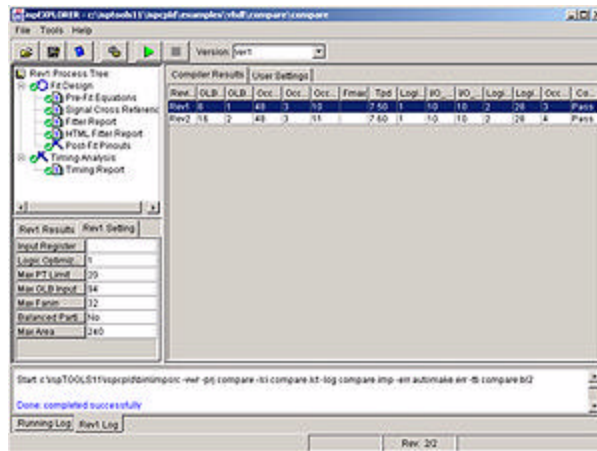
Some of the Lattice devices contain PLL circuits. You can instantiate PLL in your design source files to access PLL functions. You can also use the Constraint Editor to set PLL attributes.

HSI

Some of the Lattice devices contain HSI circuits. You can instantiate HSI modules in your design source files to access HSI functions. You can also use the Constraint Editor to set HSI attributes.

The ispEXPLORER

The ispEXPLORER lets you run multiple passes of your design using different combinations of Fitter/Optimizer settings and critical timing constraints to achieve the best solution. Results are summarized in a single spreadsheet and detailed reports for each run are accessible.



Overview of the ispEXPLORER

The ispEXPLORER software helps you select the best constraint settings for your design by letting you run the design repeatedly with varied settings and compare the results. The software gives you a single spreadsheet summary of results and settings after completion, making it easy to compare one group of settings with another.

The ispEXPLORER supports ispMACH 4000B/C, ispMACH 5000VG, and ispMACH 5000VE devices only.

Use predefined or customized settings.

You can create a single run or multiple runs using different combinations of settings. When you accept the settings for the predefined files, which is the initial default setting, the software automatically creates a run for each of the .lci files using the predetermined settings. When you use customized settings, you can change each of the values. You can also select multiple values for each setting and have the software keep running the design until it has exhausted all possible combinations among the selected values or until it has reached a specified run stop threshold.

Create multiple versions of design runs.

You can create multiple versions with different runs. Each time you click the Start button, you have the choice of creating a new version or overwriting an existing version. When you create a new version, the software creates a new version directory inside your project folder and copies the design files into this directory. It generates new constraint files based on the current project constraint file, plus the new settings you select, and runs the design using these settings. The results and settings for each run are then saved in subdirectories of the new version directory.

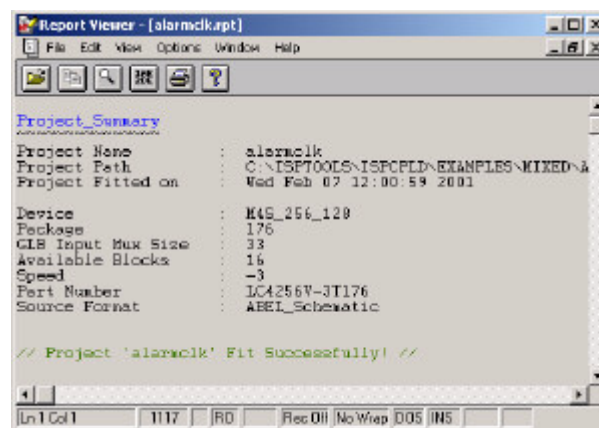
Save the best constraints for your project.

After experimenting with different settings to determine which constraint settings are best for your design, you can save the constraints by using the Save Setting pop-up command for a specific run. When you save these constraints in ispEXPLORER, the software updates your project, replacing only the constraints that you have changed; it does not replace the .lci files.

The Report Viewer

You can use the Report Viewer to view, but not edit, the various report files generated by the ispLEVER software. These reports include:

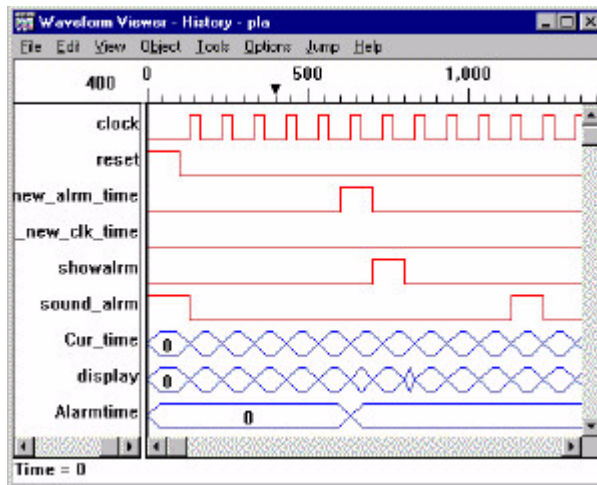
- Pre-fit equations
- Signal cross reference
- Fitter
- Timing simulation files
- Compiler listing
- Compiled equations



Design Verification

The Waveform Viewer

The Waveform Viewer displays the results of logic simulation. The nets whose waveforms are to be displayed can be interactively chosen from the schematic. Query functions can be used to trace signals to their source on the schematic. Trigger functions can be used to locate the occurrence of a specific logic event. Delays between events can be measured with markers.



Opening the Waveform Viewer

The Waveform Viewer is typically used in conjunction with a simulator. You must run the simulator before you can run the Waveform Viewer; without simulation information, the Waveform Viewer has no data to display. Therefore, you "open" the Waveform Viewer by running the simulation.

Saving and Printing Waveforms

After completing a waveform analysis, you can save the Waveform Viewer configuration using the Save command. The information saved consists of:

- Waveform names displayed
- Trigger conditions

Printing Waveforms

You can print the waveform display.

Waveform Viewer Configuration

The Waveform Viewer has several configuration variables. These variables are modified using the commands on the Options menu.

Waveform Display

A waveform is a graphic representation of the state transitions for a single input or output signal, or of a group of signals. Waveforms are displayed as traces below a horizontal time line. The name of each signal is displayed on the left side of the Waveform Viewer window. If a waveform represents a group of signals, it is displayed as a bus with its values represented in binary, octal, decimal, or hexadecimal. A group of waves may be displayed as a bus even though it might not exist as a bus on the schematic.

The most fundamental operation in the Waveform Viewer is adding waveforms to the display. Once waveforms are displayed, they can be moved, deleted, copied, and converted to bus format using the procedures described in the following sections.

Adding Waveforms

There are several ways to cause Waveforms to be displayed in the Waveform Viewer. The standard method is to use the Probe Item command of the Hierarchy Navigator and select the desired nets and busses. You can also add waveforms when the Waveform Viewer starts, if it finds a configuration file (.wav file) saved from a previous session. Waveforms may also be added explicitly by using the Show command.

Finding the Signal You Want

In the Show Waveforms dialog box, the large list box at the left and the control button above it simplify navigating the hierarchy to find the signals you want. The list box initially displays the top level of the hierarchy. Clicking the **Push** button displays the hierarchical level (if any) below the top level.

To move to a lower hierarchical level, highlight that level in the list box, and then click the **Push** button. (If you are already at the lowest level, the button is relabeled Pop, since you can only move upward in the hierarchy.) A display line below the list box shows the full hierarchical path of the level you're currently on.

All signals at a given level are shown in the right list box. To add a signal to the display, click its name, and then click the **Show** button. (Or just double-click the signal's name.) The signal is immediately added at the bottom of the Waveform Viewer display.

The waveform display can contain up to 256 waveforms. Use the vertical scroll bar to select the waveforms to view.

Using the Probe Item Command

If there is a schematic for the design, the Probe Item command is the easiest way to add waveforms to the display. Click the desired net in the Hierarchy Navigator, and the waveform for that net is added to the display. Buses from the schematic can be probed, but the bus must be probed at the highest level at which it exists in the hierarchy.

The Probe Item command is available only when the Waveform Viewer is used with the Hierarchy Navigator.

Duplicating Waveforms

The Duplicate command copies waveforms. The Duplicate command is most often used to add copies of global signals, such as clocks, to have a reference signal near the event being examined. The original waveform remains in the display.

Changing Waveform Locations

You can move a waveform from one display location to another using drag-and-drop techniques.

Hiding Waveforms

The Hide command hides waveforms from the display without deleting them from the data files.

Creating Bus Displays

You can create a bus display of two or more signals, whether or not they are related.

Expanding Bus Displays

You can use the Expand Bus command to divide a bus into its constituent components as reference signals only; you cannot edit these individual signals.

Changing the Bus Radix

The Waveform Viewer displays Bus values on the bus waveforms, and on the prompt line if a bus is selected. You can change the bus radix using the Bus Radix command.

Moving Around

Once the waveforms are displayed, there are several ways to manipulate the waveform display area.

View Commands

The View commands change the horizontal time dimension. Different time segments of the displayed waveforms can be viewed.

Zoom In	Increases the horizontal magnification each time it's executed. You see a shorter time segment in more detail. You can also drag around any part of a waveform to view it in more detail.
Zoom Out	Reduces the current magnification each time it's executed. You see a longer time span with less detail.
Pan	Slides the current viewing window across the waveforms. The point you click becomes the new center point of the display. The magnification does not change.
Full Fit	Clicking inside the window fills the display with the full time span of the displayed waveforms. Two options are then available:

Click the location you want to see in more detail. This returns the window to the previous magnification and pans the view to the selected point.

Drag the mouse to form a box around the area you want to zoom in on. The magnification is adjusted to display that area.

Scroll Bars

The horizontal scroll bar under the waveform display positions the time scale. The vertical scroll bar controls the position within the set of visible waveforms.

Moving the Query Cursor

Several commands from the Jump menu move the query cursor.

Tick Left, Tick Right	Moves the cursor left and right by one small tick mark. They are useful for slowly scanning a waveform, or for accurately positioning the cursor at an event. The time represented by one small tick mark changes as the scale is changed with the View commands.
10 Left, 10 Right	Moves the query cursor to the left or right by one large tick mark (equal to 10 small tick marks). The time, represented by a large tick mark, changes as the scale is changed with the View commands.
Time=0, Time=End	Jumps to the beginning or end of the waveform.
To Marker	Jumps to the current marker position.
To Time	Lets the display centered on the time you specify.
Next Change	Jumps to the next change in signal polarity.
Next Trigger	Jumps to the next trigger point.

Marking Your Spot

The Place Marker command inserts a marker (a dashed, colored, vertical line) when you click the Query cursor on a waveform. The marker is useful as a reference point for measuring times between events. The time difference between the time at the Query cursor and the time at the marker is displayed on the prompt line.

Jumping to Events

Events are logic-level changes. A change in any signal in a bus is considered an event on that bus. Timing measurements are usually made between events.

Several commands in the Waveform Viewer make it easier to find events and align the cursor to events. These commands are especially helpful when the

display is zoomed out and the resolution is too low to accurately position the cursor.

The Next Change command moves the query cursor to the next event on the selected waveform. It's commonly used to measure the time difference from one event to another.

Setting Signal and Bus Triggers

A *trigger* is an event that meets some specified criteria. The Set Trigger command lets you apply one of several conditions to one or more waveforms. A *trigger event* occurs when all the conditions on all the waveforms are met. You can locate a highly specific event by applying these criteria to several waveforms.

Trigger conditions on signals may be based either on the current state or on a transition from one state to another. The condition may be that the signal is High, Low or Unknown, or that the signal is at a Change in level, a Rising Edge or a Falling Edge.

A trigger condition on a bus is based on the state of each signal in the bus. The trigger condition for a bus has one character for each signal in the bus. The character can be 0, 1, or X ("don't-care"). Any state matches an X.

You can mix signal and bus triggers. The Waveform Viewer displays all active triggers in a list on the Trigger dialog box. When you set the triggers the way you want them, close the Trigger dialog box.

The current state of the trigger conditions is normally displayed on the prompt line. The trigger state is sometimes briefly overwritten by status messages from commands.

Analysis Techniques

This section explains the waveform-analysis commands. You might find it easier to use their accelerator keys than to select them from the menus.

Interaction with the Hierarchy Navigator

The Find Item command from the Hierarchy Navigator locates the part of the circuit driving a particular waveform. The Hierarchy Navigator automatically displays the appropriate schematic. The net associated with the waveform is highlighted.

This command is useful when you find an interesting event in the waveform display and want to locate the source of the event on the schematic. The Find Item command works only with the Hierarchy Navigator.

The Query command highlights the net associated with the currently selected waveform. If the query window in the Hierarchy Navigator is already open, its contents change to reflect the latest net queried with the Query command in the Waveform Viewer.

The Probe Item command adds waveforms to the Waveform Viewer display when you probe a net in the schematic.

Displaying Simulation Values on a Schematic

The logic values determined during simulation are displayed on the schematic loaded in the Hierarchy Navigator. As you click the query cursor at different points along the time line, the logic values on the schematic change to those for that simulation time. All logic values are displayed and updated, not just those for waveforms in Waveform Viewer's display.

The logic values are displayed on the schematic in two ways

- A small colored square is attached to any probed symbol nodes on the schematic. The color of the square indicates the logic value. (The default value is yellow for high, blue for low.) These colored squares are useful when the schematic is displayed at a low magnification and the text is too small to be read.
- Inside the small colored square is the text representation of the logic value. The text value is 0, 1, X (unknown), Z (high impedance) or the value of a bus.

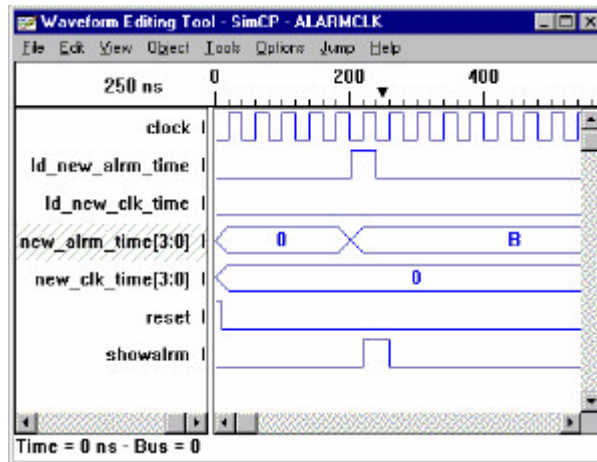
Viewing Reports

The View Report command reads error information from a file and displays the errors interactively, one error to each line. Clicking a line moves the waveform display to the corresponding error. If the View Report command is used with the Hierarchy Navigator, the schematic is displayed and the pin driving the net with the problem is highlighted.

The Waveform Editor

The Waveform Editor lets you graphically create a test stimulus file by clicking and dragging with the mouse. You see exactly what each waveform will look like, as well as its timing relationship to all the other waveforms.

The Waveform Editor can be used as a standalone tool or interactively with the Hierarchy Navigator.



Bus Pulses

Bus waveform pulses are shown as elongated hexagons. Bus pulses can have any value that can be represented by the number of bits in the bus. For example, any pulse in an eight-bit bus waveform can be assigned a value between 0 and 255.

Bus pulses are given default values, in sequential order: 0, 1, 2, 3, and so on. You can change any of these values at any time. Click the bus pulse to which you want to assign a new value, then type the new value in the Value edit box of the Toolbox.

The format for bus value display and entry is set in the Display Options dialog box invoked from the Options menu. If the number you entered is not displayed correctly, be sure you are entering it in the currently selected Bus Radix format.

Patterns

You may find yourself drawing the same waveform over and over again. The Waveform Editor allows you to name and define arbitrary waveforms, called *Patterns*, which you can then add to or insert in any other waveform. When you edit the Pattern waveform, all waveforms that use that Pattern change.

Patterns normally represent single-bit data. You can create a bus Pattern by adding a bit range to the name, as shown below:

```
$buspat [7:0]
```

Bus Patterns cannot be added to single-bit waveforms, or vice versa. Also, the number of bits in a bus Pattern must match the number of bits in the bus waveform to which it is added.

Hierarchical Patterns

Patterns can be defined hierarchically. That is, one Pattern can contain other Patterns, which themselves include other Patterns, and so on. This feature makes it possible to build complex Patterns from simpler "building block" Patterns.

Note: *Hierarchical Patterns cannot be defined recursively. In other words, you cannot add a copy of a Pattern to itself.*

Editing Patterns

You edit Patterns the same way as waveforms. When you change a Pattern, all the waveforms containing it change.

You can edit either the Pattern itself, or an instance of that Pattern in another waveform ("in-place" editing). If you edit the instance, the original Pattern is also modified. All changes are shown immediately on the screen.

Note: *You can do "in-place" editing, only if the waveform of the Pattern is displayed (rather than a rectangle). The Pattern Waveforms checkbox in the Display Options dialog box must be selected (or the Pattern Names checkbox must be cleared).*

Simulation Time

The Simulation Time is the total runtime for the simulation. The default value is 1,000,000 times the Times Unit value. For example, if the Time Unit selected is 0.01 ns, the default value in the Simulation Time edit box is 10,000.0 ns. You can change the Simulation Time by clicking inside the edit box and typing the value you want.

Note: *A long Simulation Time combined with short Time Units may require an excessive amount of computer time. Be sure you have selected values appropriate for your design before you begin simulation.*

Simulator Setup

The Waveform Editor uses the specifications in the [Export] section of the simulator initialization file (`simulator.ini`) to define the exported waveforms' file format.

Stimulus File Format

The target simulator determines the stimulus file format. The simulator must be specified in the `.ini` file so that the Waveform Editor knows which `simulator.ini` file to read to obtain the correct conversion and formatting instructions.

Saving Changes

The first time you use the **File > Save** command, the Waveform Editor creates two files, `project.wet` and `project.wdl`. The `.wet` file contains the names of

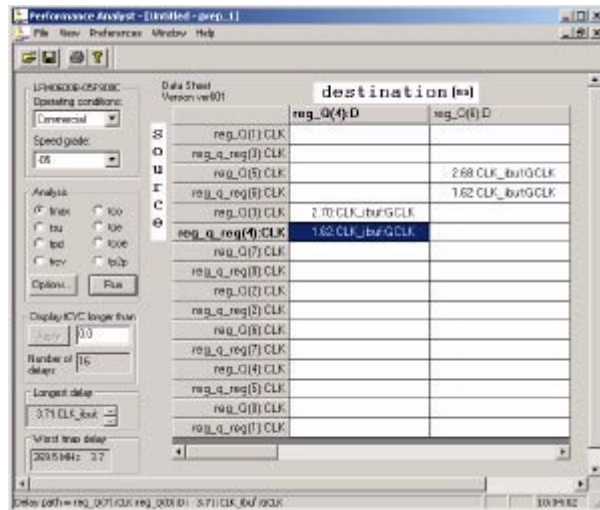
the nodes or signals you have created waveforms for. The .wdl file contains the waveforms themselves, in WDL (waveform description language) format. You can use the Save As command to save files under a different base name to create multiple stimulus files for a single project.

Caution: *You can perform an unlimited number of Undos and Redos until you save the file. At that point, the files are updated and all Undo/Redo information is lost. Do not save the file if there are still changes you want to Undo or Redo.*

The Performance Analyst

Analyst is a static timing analysis tool that lets you quickly determine the performance of designs implemented in any Lattice Semiconductor device.

Worst-case signal delays are reported in a graphical spreadsheet format that you can filter to verify the speed of critical paths and identify performance bottlenecks.



How does it Work?

Static timing analysis is the process of verifying circuit timing by totaling the propagation delays along paths between clocked or combinational elements in a circuit. The analysis can determine and report timing data such as the critical path, setup/hold time requirements, and the maximum frequency.

The Performance Analyst traces each logical path in the design and calculates the path delays using the device's timing model and worst-case AC specs supplied in the device data sheet.

The timing analysis results are displayed in a graphical spreadsheet with source signals displayed on the vertical axis and destination signals displayed on the horizontal axis. The worst-case delay value is displayed in a spreadsheet cell if there is at least one delay path between the source and destination. To more easily identify performance bottlenecks, you can double-click a cell to view the path delay details.

Analysis Types

The Performance Analyst performs six distinct analysis types: fMAX, tSU, tPD, tCO, tOE, and tCOE. The first type, fMAX, is an internal register-to-register delay analysis. fMAX measures the maximum clock operating frequency, limited by worst-case register-to-register delay. The remaining five types are external pin-to-pin delay analysis. Timing threshold filters, source and destination filters, and path filters can be used to independently fine-tune each analysis.

fMAX

Maximum Clock Operating Frequency

The fMAX path trace analysis reports the worst-case fMAX (maximum clock operating frequency) for each clock in the design. fMAX is equal to the reciprocal of the worst case register-to-register delay.

The Performance Analyst reports all register-to-register delays in a spreadsheet format with clock sources displayed. You can specify which clocks the Performance Analyst reports in the spreadsheet and whether tracing is enabled through all tracing paths. When there are no register paths in the design, the Run button is disabled and the spreadsheet is empty.

The Performance Analyst does not attempt to report external fMAX, because it cannot make assumptions about the arrival time of signals driving Lattice CPLD device inputs and the tSU of devices driven by Lattice CPLD device outputs. Therefore, it is up to you to determine the external fMAX based on the operating requirements of the system.

Default fMAX Path Trace

The default fMAX path starts at the source register clock input and traces through the clock-to-output path of the register, through any number of levels of combinatorial logic (through internal feedback only), to the D, T, or CE inputs of the destination register, including destination register setup time (tSU).

The Performance Analyst assumes that the same clock signal and the same edge of the clock signal clock the source and destination registers. However, delays are calculated when the source and destination registers are clocked by two clock signals or by different edge of the same clock signal. In the first case, the delay obtained is actually the setup time of the destination register through the clock-to-output path of the source register. In the second case, the actual fMAX will be half of what is calculated by the tool.

tSU

Setup Time

The tSU path trace analysis reports setup and hold time for data and clock enable signals with respect to a clock edge, or the register recovery time from asynchronous S/R inputs. You can specify whether tracing is checked at the register's D/T, CE or S/R inputs.

Default tSU Path Trace

This data path starts at an input pin and then traces through any number of levels of combinatorial logic to the D, T or CE inputs of a register. The internal tSU of the register is added to the delay path. The value of the internal tSU is dependent on the register being clocked by a global clock or product term clock.

Path Endpoints for tSU

$tSU = (\text{longest_data_path_delay}) - (\text{shortest_clock_path_delay}) + (\text{internal_setup_time})$.

$tHD = (\text{longest_clock_path_delay}) - (\text{shortest_data_path_delay}) + (\text{internal_hold_time})$.

For simplicity, in the Timing Analysis spreadsheet tHD will be shown as a "0" if the calculation is negative, regardless of its value. However, the exact hold time can be observed on the Expanded Delay Path window, which is opened by double clicking in the spreadsheet cell.

Register D/T Inputs

Reports tSU / tHD at Register data-input (D/T).

Register CE Inputs

Reports tSU / tHD at Register Clock Enable (CE).

tPD

Propagation Delay Time

The tPD path trace analysis reports input pin to output pin delay of combinatorial signals. You can specify whether reporting is enabled for paths traced through asynchronous register inputs and transparent latches.

Default tPD Path Trace

This path starts at an input pin and traces through any number of levels of combinatorial logic, through the data path of the output buffer, to the output pin.

tCO

Clocked Output-to-Pin Time

The tCO path trace analysis reports clock-to-out delay starting from the primary input, going through the clock of flip-flops or gate of latches, and ending at the primary output. You can specify whether reporting is enabled for paths traced through asynchronous register inputs, ripple clocks, or data-input of transparent latch.

Default tCO Path Trace

This path starts at an input pin and traces through any number of levels of combinatorial logic to the clock pin of a register. Tracing continues through the clock-to-output path of the register and through any number of levels of combinatorial logic, through the data path of the output buffer to the output pin. Only a single register clock-to-output delay exists in this path.

When tracing input latch gate to output delays, the path starts at the pin, traces through the gate-to-output path of the latch and through any number of levels of combinatorial logic, through the data path of the output buffer to the output pin. Only a single latch gate-to-output delay exists in this path.

Paths are not reported if they trace through asynchronous register set/reset inputs, ripple clocks, output enable paths, or transparent input latches.

tOE

Output Enable Path Delay

The tOE path trace analysis reports the input pin-to-output enable path delay starting from the primary input, through the Enable of output buffers, ending at the primary output. You can specify whether reporting is enabled for paths passing through the asynchronous register inputs or data-input of transparent latches.

Default tOE Path Trace

This path starts from the primary input pin and traces any number of levels of combinatorial logic, through the Enable of output buffers, to the primary output.

tCOE

Clock to Output Enable Time

This path trace analysis reports the input-clock-to-output-enable path delay starting from the primary input, going through the clock of flip-flops or gate of latches, going through the Enable of output buffer, and ending at the primary output. You can specify whether reporting is enabled for paths traced through asynchronous register inputs, ripple clocks, or data-input of transparent latch.

Default tCOE Path Tracing

This path starts from primary input pin and traces through the Register Clock, through any number of levels of combinatorial logic, to the Enable of output buffers.

Path Tracing Rules

The path tracing rules are designed to let you intuitively explore many aspects of the design timing in an obvious fashion. Static timing analysis options let you specify which rules the path tracing routines follow.

Tracing Enabled Through Bi-directional Paths

This path starts at the source register clock input, traces through the clock-to-output path of the register, through any number of levels of combinatorial logic, and through the data input of the output enable buffer to the output pin. Tracing continues through the input pin and through any number of levels of combinatorial logic to the D/T/CE input of the destination register, including the destination register tSU.

Tracing Enabled Through Register Asynch S/R Inputs

When this path type is enabled, paths through register S/R inputs to their Q outputs are treated as combinatorial logic.

Tracing Enabled Through Transparent Latch D Inputs

When this path type is enabled, paths through data inputs of transparent latches are reported.

Tracing Enabled Through Ripple Clocks

When this path type is enabled, tCO of registers clocked by ripple clocks are reported.

Batch Timing

Running Timing Analysis in Batch Mode

There may be times when you want more precision and flexibility while running timing analysis than is available with the Performance Analyst user interface. For example, on the Options dialog you can select Bi-directional path tracing as either "on" or "off." However, this selection applies to all Bi-directional I/Os in design. There is no way to select an individual one or a partial set.

In addition to using the graphical user interface to run timing analysis, you can run the Performance Analyst in "batch mode." This feature is called the *Batch Timer*. The Batch Timer executes a user-predefined command file and puts the result into a log file.

Batch Commands

There are four groups of commands supported in the Batch Timer:

- Set Operations
- Path Tracing
- Report
- Switch Control

Set Operations

```
SHOWSET PI | PO | CLOCK | GATED | LATCH | SR | OE
| CE | CLKIN | U_STOP | U_PASS|...
```

```
ADDSET      setname      added_set/pin
```

```
REMSET      setname      removed_set/pin
```

Set operation commands let you check pins located in the Timer set. These include **PI** (primary input), **PO** (primary output), **CLOCK** (clock of FF/L), **GATED** (D of FF/L), **LATCH** (D of Latch), **SR** (set/reset), **OE** (output-enable), **CE** (clock-enable), **CLKIN** (global clock), **U_STOP** (user-defined-stop for ignored false path), **U_PASS** (user-defined-pass for transparent latch), and etc.

You can ignore any path passing through a particular point by putting it into **U_STOP**. The command is **ADDSET U_STOP macrocell_stopped**.

Path Tracing

```
LONGEST set2set | pin2pin
SHORTEST set2set | pin2pin
```

Path tracing commands let you get paths between any two sets or any two points. The path will be expanded and reported as each step in detail if it is pin2pin mode. For example:

```
LONGEST p1 p0
LONGEST input_a reg_b.d
```

Report

```
REPORTSET [maxpath=n] [threshold=m] [excel=1|0]
REPORT file_name
```

Report commands let you specify the number of paths in each section that need to report, can specify the delay threshold so that only paths longer than the threshold needs to report and can specify the report in Excel-mode, i.e., tab' is inserted.

The Batch Timer can generate the Timing Report under the file name specified.

Switch Control

```
SWITCH passBI | passSR | passCLK |
passLatch 1|0
```

User can turns the switches ON (1) or OFF (0) to select the path-tracing enable through Bi-directional, Set/Reset, Transparent Latch, or Ripple Clock trace paths. For example:

```
SWITCH passCLK 1
```

Batch File Example

The following is an example of a batch timing analysis file. Comment lines start with '//'. The file is not case sensitive. The example design is called "alarmclk."

Batch Command File Example

```
// This is a sample batch command file for the
"alarmclk" design.
longest pi po
showset clock
longest clock sound_alm
```

```
switch passCLK 1
longest clock sound_alm
report alarmclk.trp
```

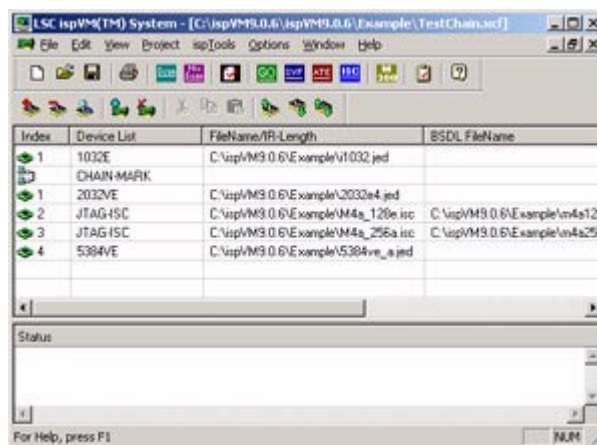

Device Programming

The ispVM System

The ispVM™ System software (ispVM) supports both sequential and concurrent (turbo) programming of all Lattice devices in a PC environment. The software scans device chains automatically, and its built-in file manager lets you browse for any required JEDEC, BSDL, ISC Data, or SVF files.

The ispVM software enables In-system Configuration (ISC) of non-Lattice devices that are compliant with IEEE 1532, allowing you to program chains from multiple vendors. The software also supports devices that are compliant with IEEE 1149.1.

Non-Lattice devices can also be programmed through the vendor-supplied SVF files.



Overview of ispVM System Software

The ispVM software processes designs through a combination of XCF and BSDL structures. Most of Lattice's devices use the IEEE 1149.1-1993 Boundary Scan Test Access Port (TAP) as the primary interface for in-system programming. Additionally, the software programs non-Lattice designs that are compliant with IEEE 1532 and supports those that are compliant with IEEE 1149.1.

Lattice Designs

Once a design has been compiled to a JEDEC file and device programming is necessary, the fuse map data must be serially shifted into the device along with the appropriate addresses and commands. Traditionally, programmable logic devices have been programmed on PLD/PROM programmers, so the programmer generates all the programming signals and algorithms. The programmer also generates the external super voltage or high voltage required by non-ISP devices (typically 12-14 volts). This super voltage requirement is

one of the reasons dedicated programmers are used to program conventional PLDs.

With In-System Programmable (ISPTM) devices, the ISP programming super voltage is generated within the device. Lattice ISP devices use nonvolatile Electrically Erasable CMOS technology and require only TTL-level programming signals. An integrated state machine controls the sequence of programming operations, such as identifying the ISP device, shifting in the appropriate data and commands, and controlling internal signals to program and erase the Electrically Erasable cells in the device. Programming consists of serially shifting the logic implementation stored in a JEDEC file into the device along with appropriate addresses and commands, programming the data into the Electrically Erasable CMOS logic elements, and shifting the data from the logic array out for device programming verification.

Designs Compliant with IEEE 1532

The ispVM software provides programming support for in-system configurable (ISC) non-Lattice devices that are compliant with the IEEE 1532 standard. This allows you to program ISC devices from multiple vendors. It also allows you to program them with your Automated Test Equipment (ATE), saving you a step in the manufacturing flow.

The ispVM software includes a BSDL processing unit, which processes BSDL files with the IEEE 1532 extension. The ISC compiler reads the BSDL file, which contains your programming algorithm, and your ISC data file. It performs both syntax and semantic checking and compiles the information into an executable structure. If an ISC data file is not provided, then the software compiles only the BSDL file.

Designs Compliant with IEEE 1149.1

Non-Lattice devices that are compliant with IEEE 1149.1 can be bypassed once their instruction register length is defined in the chain description. Programmable devices from other vendors can also be programmed through the vendor-supplied SVF file.

Programming Basics

To successfully program devices in-system, there are a few simple requirements that must first be met. The first of these requirements is that the devices on the board must be correctly connected into an 1149.1 scan chain. This scan chain can be used for either programming or testing the board. To program using the ispVM™ System software, a description of the scan chain needs to be developed. This description is called a chain file and contains basic information about all of the devices in the chain. For the Lattice devices, this includes the device type, the operation to be performed and the JEDEC file, if required by the operation. Additional information in the chain file can include the state of the I/O pins during programming along with security requirements. If non-Lattice devices are present in the chain, the instruction register length is required for these devices. The instruction register length can be found from the BSDL file or the SVF file for the device.

Another requirement for successful programming is thoughtful board design. The signals used in a scan chain (TCK, TMS, TDI, and TDO) rarely operate as fast as the data path signals on the board. However, correct board layout methodologies such as buffering for large chains, termination resistors, etc. are required to ensure trouble-free operation. Some Lattice devices have some additional pins (TRST, ENABLE, ispEN, bscanEN, EPEN and TOE) that can affect boundary scan programming and test if not taken care of properly. These board layout methodologies are described later in this document.

After all of these requirements have been met, it should be relatively straightforward to program any number of devices on a board. This programming can be done using a PC, with either a Lattice ispDOWNLOAD cable with the 8-pin AMP connector or the 10-Pin JEDEC connector attached to the board, and a board test system.

JTAG Scan Chains

A scan chain can include any IEEE-1149.1 compliant, programmable or non-programmable device or any IEEE 1532 programmable device. It can also include any programmable devices that are compatible with IEEE-1149.1 but do not have a boundary scan register. This is a decision that should be made based on the test methodology being employed for the board. If the test methodology employed is the traditional bed-of-nails approach used on board test systems, all the devices can be included in the same chain.

All scan chains use the simple four-wire TAP. The TCK and TMS pins are common to all devices included in the chain. TDI and TDO are daisy-chained from one device to the next. The input to the chain is TDI and the output from the chain is TDO. A diagram demonstrating a simple scan chain is shown in Figure 1.

Programming Algorithm Basics

Programming a CPLD is similar to programming any memory device such as an EPROM or FLASH memory. The device can be thought of as an array that is programmed one row at a time. The programming information is provided to the software in the form of a standard JEDEC file that needs to be converted into the row and column fuse map data. Before an EEPROM device can be programmed, it first has to be erased. After the device has been erased, the programming data can be loaded and the device programmed. After the device has been programmed, it will be verified by reading out the data in the device and comparing it against the original.

Figure 2 shows the basic programming flow for the device. It does not include JEDEC file data conversion into fuse map data, as it assumes that has already been done. This programming flow will be the same regardless of the programming hardware used. The primary difference between programming on different hardware platforms (PC vs. Workstation) is the type of hardware (parallel port vs. serial port) and the driver associate with it.

Note 1: *Although it is not necessary, you should always perform a reset before and after programming a device.*

Note 2: *If the device will not be programmed in-circuit (i.e. via a cable or using an embedded processor), then it is not necessary to preload or save the I/O states.*

Programming Times

The time it takes to program a device can often be a determining factor of where in the manufacturing process a device, or group of devices, is programmed. A board test system costing hundreds of thousands of dollars to purchase and costing as much as one dollar per minute to operate can be an expensive alternative for programming if programming times are too long. In many instances, it is more cost-effective to buy a couple of PCs and program the devices using these much cheaper systems.

The time it takes to completely program a device is based on the time it takes to first erase the device, then program each row in the device, and then finally to verify the device. The erase time for all devices is between 100 and 200ms. A single row is programmed in 10 to 50ms, depending on the device. The Verify process is the quickest of the required steps in the programming sequence and is mainly dependent on the time required to shift the verify data out of any given device.

To minimize the total programming time of a daisy chain of ISP devices, a programming method called Turbo ispDOWNLOAD™ can be used to program all the ISP devices in the chain concurrently. Turbo ispDOWNLOAD allows any number of ISP devices to be programmed at the same time. When programming a chain concurrently, the chain can be programmed in the time it takes to program only the largest device plus some extra time in order to shift instructions and data for multiple devices. For example, a chain of three devices with programming times of 10, 7, and 7seconds can be programmed with Turbo ispDOWNLOAD in a total of about ten seconds (the time it takes to program the largest device). Serially, the programming time would be 24 seconds for all three devices. Turbo ispDOWNLOAD is incorporated into the ispVM software. This valuable feature of Lattice device technology is not available with many other ISP CPLD device technologies.

The benefit of minimal programming times will be much more obvious on board test systems, because they are included as a part of the test program and are running at the fastest speed possible. Additionally, there is no translation needed to or from JEDEC formatted data as this has already been done by the ispVM System software.

USERCODE

User-programmable identification can ease problems associated with document control and device traceability. The ispLSI 1000EA, 2000E, 2000VE, 2000VL, 2000V, 5000V, 8000/V, MACH 4/A, ispGDX/A, ispGDXV, ispMACH 4000, and ispMACH 5000 families contain a 32-bit register accessible through the optional IEEE 1149.1 USERCODE instruction. This user-programmable ID register is basically a user's "notepad" provided in electrically erasable (E²) cells on each device.

In the course of system development and production, the proliferation of PLD architectures and patterns can be significant. To further complicate the record-

keeping process, design changes often occur, especially in the early stages of product development. The task of maintaining which pattern goes into what device for which socket becomes exceedingly difficult. Once a manufacturing flow has been set, it becomes important to "label" each PLD with pertinent manufacturing information, which is beneficial in the event of a customer problem or return. A USERCODE register is incorporated into ISP devices to store such design and manufacturing data as the manufacturer's ID, programming date, programmer make, pattern code, checksum, CRC, PCB location, revision number, and/or product flow. This assists users with the complex chore of record maintenance and product flow control. In practice, the user-programmable USERCODE register can be used for any of a number of ID functions.

Within 32 bits available for data storage, users may find it helpful to define specific fields to make better use of the available storage. A field may use only one bit (or all bits), and can store a wide variety of information. The possibilities for these fields are endless, and their definition is completely up to the user.

Even with the device's security feature enabled, the USERCODE register can still be read. With a pattern code stored in the USERCODE register, the user can always identify which pattern has been used in a given device. As a second safety feature, when a device is erased and re-patterned, the USERCODE identification is automatically erased. This prevents any situation in which an old USERCODE might be associated with a new pattern.

It is the user's responsibility to update the USERCODE when reprogramming. It should be noted that the USERCODE information will not be included in the fusemap checksum reading.

Loading of the USERCODE instruction makes the USERCODE available to be shifted out in the Shift-DR state of the TAP controller. The USERCODE register can be read while the device is in normal functional operation, allowing the device to be scanned while operating.

I/O States During Programming

During a programming cycle, all ispJTAG devices default to having their I/Os tri-stated. In most situations, this probably will be acceptable and will not cause any problems. However, there are situations that arise where it may cause some contention. Through the boundary scan cells of Lattice devices, ispVM offers the capability of setting all I/O pins to a state of "1", "0", HIGHZ, or don't care, and to set the state of each I/O pin individually.

Programming Hardware

All ISP programming specifications, such as the programming cycle and data retention, are guaranteed when programming ISP devices over the commercial temperature range (0 to 70 degrees C). It is critical that the programming and bulk erase pulse width specifications are met by the programming platform to insure proper in-system programming. The details of device programming are transparent to the user if Lattice ISP programming hardware and software are used.

PC Hardware

Programming is most commonly done on a PC through a download cable attached to the parallel port using the ispVM System software.

ispDOWNLOAD Cable

The ispDOWNLOAD cable is designed to facilitate in-system programming of all Lattice ISP devices on a printed circuit board directly from the parallel port of a PC. The ispVM System software generates programming signals directly from the parallel port of a PC, which then pass through the ispDOWNLOAD cable to the device(s). With this cable and a connector on the board, no additional components are required to program a device. Refer to the ispDOWNLOAD Cable Data Sheet for more detailed specifications and ordering information.

Hardware design considerations for new boards include whether the hardware designer will be using boundary scan test operations or low voltage (3.3V) devices. In a system using 3.3V ISP devices, the ispDOWNLOAD cable version 2.0 should be used. The cable operates with either a 3.3V or 5V Vcc source.

Note: *If you are using the ISP Engineering Kit Model 300, you must use the ispDOWNLOAD cable Version 2.0, not version 1.*

ISP Engineering Kit Model 300

The ISP Engineering Kit Model 300 provides designers with a quick and inexpensive means of evaluating and prototyping new designs using Lattice devices when compared to a standalone programmer. This kit is designed for engineering purposes only and is not intended for production use. The kit programs devices from the parallel printer port of a host PC. By connecting a system cable (included) from the host PC to the ISP Engineering Kit with the proper socket adapter, a device can be easily programmed using the ispVM System. See Figure 3.

Note: *If you are using the ISP Engineering Kit Model 300, you must use the ispDOWNLOAD cable Version 2.0, not version 1.*

Programming Software

The ispVM System software supports programming of all Lattice ISP devices in a serial daisy chain programming configuration in a PC environment. Any required JEDEC, ISC Data, or BSDL files are selected by browsing with a built-in file manager. This software supports both sequential and concurrent (turbo) programming of all Lattice devices and programming of ISC devices compliant with IEEE 1152. Any non-Lattice devices that are compliant with IEEE 1149.1 can be bypassed once their instruction register length is defined in the chain description. Using ispVM, programmable devices from other vendors can be programmed through the vendor supplied SVF file.

Programming on a Board Test System

Programming on a board test system is made possible by using the ATE feature to generate the necessary programming files needed for the different platforms. The platforms supported include GenRad, Agilent (HP), and Teradyne board test systems. A generic JEDEC vector format is generated to help support any ATE not directly supported.

Programming on JTAG Test Systems

JTAG test systems differ from traditional board test systems in their basic test methodology. These systems use only the four wire JTAG TAP to perform any interconnect and functional tests. A simple language has been developed to interface with the TAP and is used by most major JTAG test system vendors. This language is known as the Serial Vector Format (SVF) and is supported by the ispVM System software.

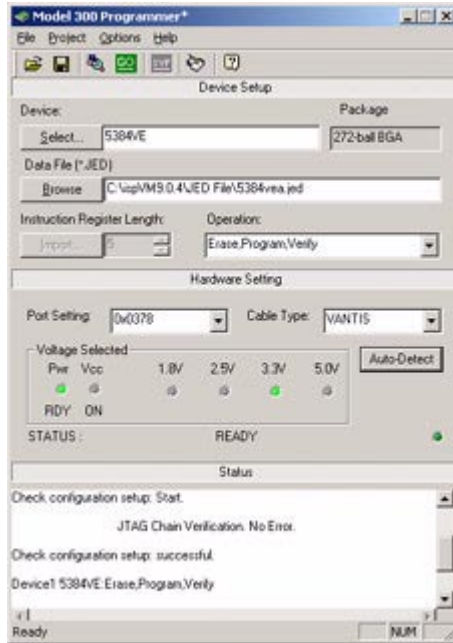
Embedded Programming

The ispVM Embedded source code is available for programming devices in an embedded or customized environment. The programming source code is written in ANSI-standard C language, which can be easily incorporated into an embedded system or tester software to support programming of ISP devices. This code supports such common operations as Program, Verify, Erase, and Secure. After completion of the logic design and creation of JEDEC or ISC Data files, the ispVM System software creates the data files required for in-system programming on customer-specific hardware: PCs, testers, or embedded systems.

HDR	(Header Data Register) Specifies a header pattern, which is placed at the beginning of subsequent DR, scan operations.	Supports TDI keyword only. SMASK, TDO and MASK are not supported.
HIR	(Header Instruction Register) Specifies a header pattern, which is placed at the beginning of subsequent IR, scan operations.	Supports TDI keyword only. SMASK, TDO and MASK are not supported.
RUNTEST	Forces the 1149.1 bus to the RUN_TEST/IDLE state for a specified number of clocks.	Full Support
SDR	(Scan data Register) Performs an 1149.1 Data Register scan.	Supports TDI, TDO and MASK keywords. SMASK is ignored.
SIR	(Scan Instruction Register) Performs an 1149.1 Instruction Register scan.	Supports TDI, TDO and MASK keywords. SMASK is ignored.
STATE	Forces the 1149.1 bus to a specified stable state.	Supports only single target state. The debugger does not allow the user to specify custom traverse path.
TDR	(Trailer Data Register) Specified a trailer pattern, which is appended to the end of subsequent DR, scan operations.	Supports TDI keyword only. SMASK, TDO and MASK are not supported.
TIR	(Trailer Data Register) Specified a trailer pattern which is appended to the end of subsequent IR scan operations	Supports TDI keyword only. SMASK, TDO and MASK are not supported.

The Model 300 Programmer

The ISP Engineering Kit Model 300 programmer is an engineering device programmer that supports prototype development by allowing single-device programming directly from a PC. The Model 300 programmer supports all JTAG devices produced by Lattice, with device Vcc of 1.8, 2.5, 3.3, and 5.0V.



Overview

The Model 300 programmer is launched through the ispVM System software. The following describes the Lattice devices and socket adapters that Model 300 supports, plus connection and application requirements.

Device Support

The Model 300 programmer supports all ispLSI, MACH4, MACH5, ispM4As, M5/1s, ispGDX, ispGDS, and ispGAL devices, with device Vcc of 1.8, 2.5, 3.3, and 5.0V.

Socket Support

The Model 300 supports the adapter pinout configuration of the existing Model 100 programmer. The Model 300 is compatible with existing Lattice socket adapters and is also compatible with the Lattice's approved 28-pin DIP third-party programming socket adapters.

Connector Support

A 10-pin JTAG connector is available on the Model 300. It can be driven by the Lattice Version 2 or later download cable, or it can be driven by the Vantis download cable.

Power Supply Support

The Model 300 programmer requires 9V AC or DC at 1A minimum, to provide power for the programmer itself and for programming the target device.

Programming Software Support

Model 300 supported application version 1.0 is used. It can be launch by ispVM System 9.0.x.

Special Features

With the exception of programmer power and a set of calibration control switches, the Model 300 programmer is controlled entirely by the programming software. When you select a device, you also select the adapter configuration and target device Vcc. LED buttons indicate the power status and the selected Vcc level.

Calibration Control Switches

A set of four slide DIP switches within the Model 300 provides override capabilities. These are LOCAL, LSB, MSB, and VCC ON.

LOCAL

Off This is the normal operation, under software control. LSB and MSB switches are disabled.

On Programming software is connected to TAP of M4A5-64/32 programmer control IC. This allows on-board programming of the control IC. LSB and MSB switches are enabled.

LSB, MSB, VCC ON

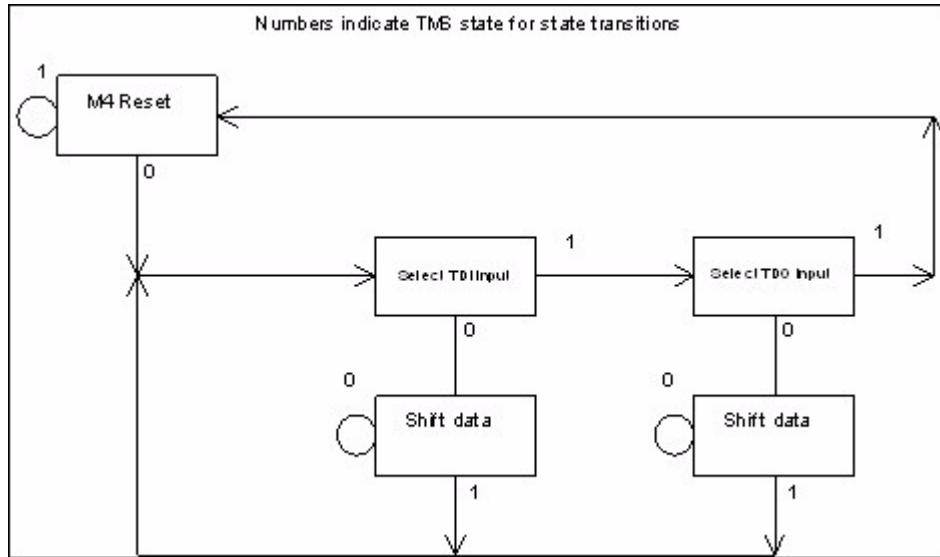
When LOCAL is **On**, LSB and MSB select the Vcc level at the programming adapter sockets, as indicated by the LED indicators. Switch VCC ON turns on the programming socket adapter Vcc for measurement at the DVCC test point, while the selected Vcc level is adjusted at R16, R18, R20, or R22. Switch VCC ON is independent of switch LOCAL.

Software Control

The Model 300 programmer is controlled by the TRST, ENABLE (ispEN), TMS, TDI, and TCK signals from the download cable. When in normal operation mode, the Model 300 programmer controls IC channels TMS, TDI, TCK, and TDO to/from the programming adapter socket. TRST and ENABLE control the programmer's operation mode.

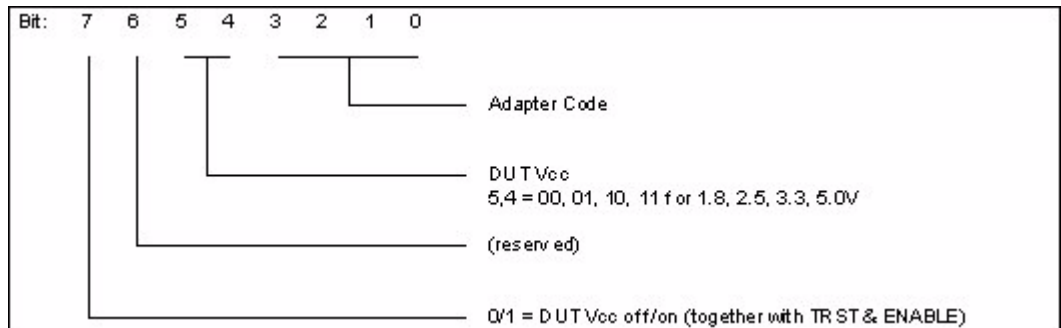
TRST	ENABLE	Description
0	0	DUT is target of JTAG signals, DUT Vcc is off
0	1	DUT is target of JTAG signals, DUT Vcc is on if Control Bit 7 = 1
1	0	Control IC TAP is target of JTAG signals, DUT Vcc is off
1	1	Control IC State Machine is target of JTAG signals, DUT Vcc is off when State Machine is not in M4 Reset state

When TRST and ENABLE are high and LOCAL is Off, JTAG signals interact with the State Machine programmed into the M4A5-64/32 Control IC. Use this State Machine to write or read Control Register bits for routing the JTAG signals to the appropriate pins for the selected adapter and for controlling DUT Vcc.



When TDI input is selected, data is shifted into the State Machine from TDI, to write new data into the Control Register. When TDO input is selected, data is rotated through the Control Register to read it out on TDO. It must be shifted eight times to return the Control Register to its original pattern.

Control Register



Valid Adapter Code values are 0000, 0001, 0010, 0011, 0100, and 1100.