# Contents

**Preface** ................................................................. 9
*About This Manual* ......................................................... 9
*Finding Information in This Manual* .................................. 10
*Typographical Conventions* .............................................. 11
*Related Documentation* ................................................... 12

## 1
### Getting Started ...................................................... 13
*Library Management Use Model* ......................................... 13
  *As a Librarian* .......................................................... 13
  *As a Designer* ........................................................... 14
*Starting the Tool* ......................................................... 14
  *Library Explorer* ....................................................... 14
  *Part Developer* .......................................................... 15
  *Part Table Editor* ...................................................... 15

## 2
### User Interface of Part Developer ................................ 17
*Overview* .................................................................. 17
  *Part Developer Work Environment* ................................... 18
*Part Developer Menus* ..................................................... 19
  *File* .................................................................. 19
  *Edit* .................................................................. 21
  *View* .................................................................. 22
  *Tools* .................................................................. 22
  *Templates* .............................................................. 23

## 3
### Creating Parts .......................................................... 25
*Part Creation Methodology* ................................................. 25
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring Part Developer</td>
<td>26</td>
</tr>
<tr>
<td>Setting Up Property Defaults</td>
<td>26</td>
</tr>
<tr>
<td>Setting Up Pin Defaults</td>
<td>29</td>
</tr>
<tr>
<td>Setting Up Symbol Defaults</td>
<td>31</td>
</tr>
<tr>
<td>Setting Up Format Defaults</td>
<td>33</td>
</tr>
<tr>
<td>Setting Up Pin Interpretation Defaults</td>
<td>37</td>
</tr>
<tr>
<td>Saving Part Defaults</td>
<td>41</td>
</tr>
<tr>
<td>Loading Part Defaults</td>
<td>42</td>
</tr>
<tr>
<td>Part Creation Situations: Symmetrical, Asymmetrical, and Parts with Pins Split Across Symbols</td>
<td>42</td>
</tr>
<tr>
<td>Creating Parts</td>
<td>48</td>
</tr>
<tr>
<td>Pin Naming</td>
<td>48</td>
</tr>
<tr>
<td>Entering Part Properties and Logical Pins Information</td>
<td>49</td>
</tr>
<tr>
<td>Handling of Power and Ground Pins</td>
<td>52</td>
</tr>
<tr>
<td>Creating Packages</td>
<td>53</td>
</tr>
<tr>
<td>Creating Other Packages</td>
<td>61</td>
</tr>
<tr>
<td>Selecting a Default Package</td>
<td>63</td>
</tr>
<tr>
<td>Creating Symbols</td>
<td>63</td>
</tr>
<tr>
<td>Split Part Methodology</td>
<td>63</td>
</tr>
<tr>
<td>Creating Flat Symbols</td>
<td>64</td>
</tr>
</tbody>
</table>

## 4

### Modifying Parts

**Overview** | 65
---|---
**Modifying the Logical Pin List** | 65
| Modifying Packages | 66
| Changing Package Properties | 67
| Modifying Physical Pin Mappings | 67
| Modifying Symbols | 68
| Modifying Symbol Properties | 68
| Changing the Associated Package Type | 68
| Adding a Logical Pin to All Symbols | 70
## 5 Handling Pin Texts

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
</tr>
</tbody>
</table>

### Pin Text Methodology

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
</tr>
</tbody>
</table>

## 6 Creating VHDL Wrappers/Map Files

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
</tr>
</tbody>
</table>

### Overview

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
</tr>
</tbody>
</table>

### Creating a VHDL Wrapper/Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
</tr>
</tbody>
</table>

### Modifying a VHDL Wrapper/Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
</tr>
</tbody>
</table>

### Deleting a VHDL Wrapper/Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
</tr>
</tbody>
</table>

### Renaming a VHDL Wrapper/Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
</tr>
</tbody>
</table>

## 7 Creating Verilog Wrappers/Map Files

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
</tr>
</tbody>
</table>

### Overview

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
</tr>
</tbody>
</table>

### Creating a Verilog Wrapper and Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
</tr>
</tbody>
</table>

### Modifying a Verilog Wrapper/Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
</tr>
</tbody>
</table>

### Deleting a Verilog Wrapper/Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
</tr>
</tbody>
</table>

### Renaming a Verilog Wrapper/Map File

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
</tr>
</tbody>
</table>

## 8 Part Templates

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
</tr>
</tbody>
</table>

### Overview

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
</tr>
</tbody>
</table>

### Creating a Part Template

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
</tr>
</tbody>
</table>

### Using Part Template to Create Parts

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
</tr>
</tbody>
</table>

### Verifying a Part Against a Template

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
</tr>
</tbody>
</table>

### Extracting Templates from Existing Parts

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
</tr>
</tbody>
</table>

### From Packages

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
</tr>
</tbody>
</table>
9
Using XML

Overview
Creating Parts Using the E-tools XML Format
Post Import Issues
PIN_TYPE Property
CLASS Property
Duplicate Pins
Pins With ‘>’ or ‘<’
Partnames with ‘/’
JEDEC_TYPE
NC Pins
Symbol Interpretation
Saving Part Information in XML Format
Steps
Conversion Details
Translating Chips (Physical) View
Translating Symbol (Logical) View

10
Concept HDL Cell - Capture Part Translations

Overview
Capture to Concept HDL Conversion
Overview
Steps
Conversion Details
Examples
Concept HDL to Capture Conversion
Part Developer User Guide

Entering Part Properties and Logical Pin Information (Example of an Asymmetrical Part) ................................................................. 169
Creating Packages (Example of an Asymmetrical Part) .................................................................................................................. 171
Creating Symbols (Example of an Asymmetrical Part) ................................................................................................................. 173
Creating Asymmetrical Parts using the Expand to Slots Option .................................................................................................. 177
Collapsing Expanded Scalar Pins .................................................................................................................................................. 178

C
Creating Parts With Pins Split Across Symbols .............................................................................................................................. 179
Overview .................................................................................................................................................................................. 179
Creating Symbols ................................................................................................................................................................. 179

D
Pin Types ........................................................................................................................................................................... 185

E
Checks ................................................................................................................................................................................ 189
View Verification ........................................................................................................................................................................ 189
Instantiation and Packaging ..................................................................................................................................................... 190
Advanced View Checks .......................................................................................................................................................... 190
VHDL Compilation .................................................................................................................................................................. 190
Verilog Compilation ............................................................................................................................................................... 190
Verify with Templates ............................................................................................................................................................. 191
Property Checks ........................................................................................................................................................................ 191
Pin Load Checks ...................................................................................................................................................................... 191
Symbol Checks ........................................................................................................................................................................ 191

Index .................................................................................................................................................................................... 195
Preface

About This Manual

Part Developer and Library Explorer functionality is divided into two levels. The core capabilities are found in all versions of Library Explorer and Part Developer shipped as part of PCB Librarian and the latest packaging configurations of Concept HDL. The extended capabilities are restricted to the PCB Librarian Expert license only.

If you use the PCB Librarian Expert suite, then in addition to the ability to create library parts in a quick and intuitive manner, the expert version enables you to utilize information provided over the Internet to auto generate the library parts. Part Templates, that reflect your companies’ library creation specifications, ensure consistency across all created parts. The capability to read and write both Concept HDL and Orcad Capture parts, especially helpful when leveraging existing Orcad or Concept HDL libraries to drive new part creation. The support of XML standards not only provides a way to automatically import pin and package data, but also provides a universal format in which to store and pass on part data. Many other advanced features are provided as part of the PCB Librarian Expert and by harnessing the power of PCB Librarian Expert, engineering services departments around the world can generate library parts quickly and accurately for use with Cadence Concept HDL schematic and Cadence Allegro Layout software.

This guide assumes familiarity with a system text editor, HDL language concepts, and the following Cadence tools used to create component symbols and models:

- Library Explorer, which lets you manage component libraries.
- Part Table Editor, which lets you create part table files.
- Concept HDL, which lets you create logic designs by drawing schematics using symbols and functional blocks.
- Packager-XL, which lets you prepare your schematic for PCB layout.
- Allegro, which lets you create and manage physical layouts.
Finding Information in This Manual

Chapter 1, “Getting Started,” familiarizes you with the use model followed by librarian and designers when working with digital component libraries.

Chapter 2, “User Interface of Part Developer,” discusses the various elements of the Part Developer user-interface.

Chapter 3, “Creating Parts,” describes the part creation methodology and the steps involved in creating parts. This chapter also details the methodology followed for handling split parts and flat symbols.

Chapter 4, “Modifying Parts,” covers the steps involved in modifying packages, symbols and split symbols.

Chapter 5, “Handling Pin Text,” covers the methodology followed by Part Developer to handle pin texts in symbols.

Chapter 6, “Creating VHDL Wrappers/Map Files,” details the steps involved in creating VHDL wrappers and map files from within Part Developer.

Note: This feature is available only if you have the PCB Librarian Expert license.

Chapter 7, “Creating Verilog Wrappers/Map Files,” covers the steps involved in creating Verilog wrappers and map files from within Part Developer.

Note: This feature is available only if you have the PCB Librarian Expert license.

Chapter 8, “Part Templates,” details the steps involved in creating part construction rules and property setup through the use of part templates.

Note: This feature is available only if you have the PCB Librarian Expert license.

Chapter 9, “Using XML,” describes the process involved in reading XML data to create parts and writing out part data in XML format.

Note: This feature is available only if you have the PCB Librarian Expert license.

Chapter 10, “Concept HDL Cell - Capture Part Translations,” describes the methodology followed in converting between Concept HDL and Capture parts. This chapter also covers the steps to be followed to convert between Concept HDL and Capture.

Note: This feature is available only if you have the PCB Librarian Expert license.

Chapter 11, “Verifying Parts,” covers the various checks that you can execute on a part to ensure that the part is valid and usable.
Appendix A, “Example - Symmetrical Part Creation,” includes a complete example on how to create a symmetrical part using Part Developer.

Appendix B, “Example - Asymmetrical Part Creation,” includes a complete example on how to create an asymmetrical part using Part Developer.

Appendix C, “Creating Parts With Pins Split Across Symbols,” includes a complete example on how to create parts with pins split across symbols.

Appendix D, “Pin Types,” covers the descriptions and other details about the pin types supported in Part Developer.

Appendix E, “Checks,” describes the various checks that can be run on a part to ensure its correctness and validity.

Typographical Conventions

This list describes the syntax conventions used for tools used in the library development and management process. Where applicable, exceptions to these conventions are explicitly indicated.

<table>
<thead>
<tr>
<th>literal</th>
<th>Nonitalic or (UPPERCASE) words indicate key words that you must enter literally. These keywords represent command (function, routine) or option names.</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument</td>
<td>Words in italics indicate user-defined arguments for which you must substitute a value.</td>
</tr>
<tr>
<td></td>
<td>Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.</td>
</tr>
<tr>
<td></td>
<td>For example, command argument</td>
</tr>
<tr>
<td>[ ]</td>
<td>Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.</td>
</tr>
<tr>
<td>{ }</td>
<td>Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.</td>
</tr>
</tbody>
</table>
An ellipsis indicates that you can repeat the previous argument. If they are used with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.

argument...: specify at least one argument, but more are possible
[argument]...: you can specify zero or more arguments

A comma followed by an ellipsis indicates that if you specify more than one argument, you must separate those arguments by commas.

Courier font
Text in Courier font indicates command line examples.

Related Documentation

The following manuals give you information about other tools used during the library and part creation and management process:

<table>
<thead>
<tr>
<th>If you want to know...</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to manage digital component libraries</td>
<td>Library Explorer User Guide</td>
</tr>
<tr>
<td>How to create Part Table Files</td>
<td>Part Table Editor User Guide</td>
</tr>
<tr>
<td>How to use Concept HDL to enter schematics</td>
<td>Concept HDL User Guide</td>
</tr>
<tr>
<td>More about Concept HDL digital libraries</td>
<td>Concept HDL Libraries Reference</td>
</tr>
<tr>
<td>More about how to create and use physical layouts</td>
<td>Allegro documentation</td>
</tr>
<tr>
<td>More about properties supported by Cadence PCB design software</td>
<td>PCB Systems Properties Reference</td>
</tr>
</tbody>
</table>
Getting Started

Library Management Use Model

The PCB Librarian and PCB Librarian Expert suites provide you a set of tools to perform library creation and management tasks. Depending upon whether you are a librarian or a designer, you can use the tools to perform specific tasks.

As a Librarian

As a librarian, you can do the following.

Use Library Explorer to:

1. Create or open a build area.
2. Import reference libraries or parts you wish to modify into the build area.
3. Create any new libraries or cells.
4. Launch Part Developer to create or edit views.
5. Verify libraries.
6. Export new or modified libraries and parts to the reference library location.
7. Clean up the build library area when finished.

Use Part Developer to:

1. Create or modify symbol, package, simulation views and part table views.
2. Verify the part.

Use Part Table Editor to:

1. Create or modify a part table.
2. Verify a part table.
3. Add new parts to a part table.

**As a Designer**

As a designer, you can do the following:

1. Create a project using Project Manager.
2. Specify the project libraries while creating the project.

Now if you want to either create a new part or modify existing parts in the project libraries, you can launch the Part Developer tool. When you launch the Part Developer tool, it will display the project libraries thus enabling you to modify and create new parts only in them.

**Note:** Library Explorer will not be available on a project created through ProjectManager as library management oriented tasks like creating new libraries and categories, copying libraries etc., should only be done by a librarian.

**Starting the Tool**

**Library Explorer**

**From Project Manager**

To launch Library Explorer from Project manager:

➢ Choose *Tools > Library Explorer*.

**From the Command Prompt**

In UNIX, type the following command:

    libexp

In Windows NT, type the following command in the command prompt

    libexp
Part Developer

From Project Manager
To launch Part Developer from Project Manager:
➤ Select Tools > Part Developer.

From the Command Prompt
In UNIX, type the following command:
pdv
In Windows NT, type the following command in the command prompt:
pdv

Part Table Editor

From the Command Prompt
In UNIX, type the following command:
ptf&
In NT, type the following command in the command prompt:
ptf
User Interface of Part Developer

Overview

Part Developer is a tool for creating, editing, and verifying part data. It can be launched from Library Explorer once the user has selected an existing part or named a new part. It lets you edit the schematic symbol, physical pin data, part table data and create simulation views all within a single user interface. This allows information such as pin names and common properties to be shared across the different views of the component. With built-in error checking, Part Developer ensures that parts are consistent and error-free. Part Developer also has the ability to handle asymmetrical, large pin count and technology-independent parts. Part Developer supports automatic part data entry using XML and adherence to company guidelines using part templates.

Note: Part Developer can also be launched directly from the command line by typing pdv or from the Tools menu in Project Manager. You can launch it on any project to create or modify parts in existing libraries.
Part Developer Work Environment

Part Developer follows the standard Windows interface model. The user interface is divided into the sections as described below:

<table>
<thead>
<tr>
<th>Window Components</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titlebar</td>
<td>The title bar displays the name of the current project and the part that is open. The syntax is libraryname.partname.</td>
</tr>
<tr>
<td>Menubar</td>
<td>The menubar provides the various options that enable you to perform all the tasks involved in creating parts.</td>
</tr>
<tr>
<td>Toolbar</td>
<td>The toolbar provides you a single click method to do most of the common tasks involved in creating parts.</td>
</tr>
<tr>
<td>Client window</td>
<td>The client windows displays the various views of the part in a tree structure. You can access any view by clicking on the appropriate view.</td>
</tr>
<tr>
<td>Statusbar</td>
<td>The statusbar displays the status of Part Developer.</td>
</tr>
</tbody>
</table>
# Part Developer Menus

## File

The File menu handles the I/O operations of Part Developer. It provides the following options:

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Use the New option to create a new part, package, symbol, Verilog and VHDL wrapper and map files, and part table files.</td>
</tr>
<tr>
<td>Open</td>
<td>Use the Open option to open a part.</td>
</tr>
<tr>
<td>Save</td>
<td>Use the Save option to save all the views of the part. Pre-save checks are run automatically by the tool before saving the parts, and errors, if any, are displayed.</td>
</tr>
<tr>
<td>Save As</td>
<td>Use the Save As option to save a part in any library of the current project. You can use this mechanism to make copies of parts.</td>
</tr>
<tr>
<td>Import</td>
<td>Use the Import option to import part data from one of the following sources:</td>
</tr>
<tr>
<td></td>
<td>- Intermediate file</td>
</tr>
<tr>
<td></td>
<td>An intermediate save file is a *.exp file. This file stores partial information of a part being created. You can import such a file and continue creating the part.</td>
</tr>
<tr>
<td></td>
<td>- XML</td>
</tr>
<tr>
<td></td>
<td>- Capture part</td>
</tr>
</tbody>
</table>
### Part Developer User Guide
#### User Interface of Part Developer

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export</td>
<td>Use the Export option to export part data in one of the following formats:</td>
</tr>
<tr>
<td>Intermediate File</td>
<td>Using this option saves part information in a *.exp file. No presave checks are run when you save the part information in a *.exp file. This feature is useful when you want to intermittently save the entered information or when the part cannot be saved because of pre-save checks giving some errors.</td>
</tr>
<tr>
<td>XML</td>
<td>Using this option saves the part data in XML format.</td>
</tr>
<tr>
<td>Capture Part</td>
<td>Using this option saves the part as a Capture part.</td>
</tr>
<tr>
<td>View File</td>
<td>Use the View File option to display the package (chips.prt) or the part table file in a text editor.</td>
</tr>
<tr>
<td>Load Part Defaults</td>
<td>Use the Load Part Defaults option to load the defaults that you have set up for Part Developer in a .setup file.</td>
</tr>
<tr>
<td>Save Part Defaults</td>
<td>Use the Save Part Defaults option to save the defaults that you set up for Part Developer in the current session. The defaults are saved in a .setup file.</td>
</tr>
<tr>
<td>Note:</td>
<td>If you do not make any changes to the defaults and choose File&gt;Save Part Defaults, Part Developer writes an empty .setup file. The default values are loaded from &lt;install_dir&gt;/share/cdssetup/projmgr/cds.cpm</td>
</tr>
<tr>
<td>Change Suite</td>
<td>Use the Change Suite option to change from one product suite to another.</td>
</tr>
<tr>
<td>Exit</td>
<td>Use the Exit option to exit the software. You may be prompted to save your changes.</td>
</tr>
</tbody>
</table>
## Edit

The Edit menu handles the edit operations such as copy, paste, delete, and modify appropriate properties. The Edit menu provides the following options:

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Use the Copy option to copy a selected object to the Clipboard without removing it from the tree. This command is available only if the source or target is appropriate.</td>
</tr>
<tr>
<td></td>
<td>Copying objects to the Clipboard replaces any objects previously stored there. Use the Paste command to copy objects to another page or part.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The Copy command is available individually for a specific package, symbol, Verilog wrappers/map files or VHDL wrappers/ map files.</td>
</tr>
<tr>
<td>Paste</td>
<td>Use the Paste option to place any object stored on the Clipboard in the tree. This command is unavailable if the Clipboard is empty or if the source or target is inappropriate.</td>
</tr>
<tr>
<td></td>
<td>Pasting objects from the Clipboard does not affect the Clipboard’s contents. Use Paste to copy objects to another page or part.</td>
</tr>
<tr>
<td>Delete</td>
<td>Use the Delete option to remove the selected object from the tree without putting it on the Clipboard. This options is available only if a valid object is selected. Deleting objects does not affect the Clipboard’s contents.</td>
</tr>
<tr>
<td>Properties</td>
<td>Use the Properties option to bring up the properties dialog box of the selected item in the tree. For example, for logical pins list, it will bring up the Part Properties dialog box.</td>
</tr>
<tr>
<td>Specify Footprint</td>
<td>Use the Specify Footprint option to specify the physical properties for the part. In this, you can set up the JEDEC_TYPE and ALT_SYMBOLS properties for the part. This option is available only when you have selected a package.</td>
</tr>
<tr>
<td>Logical Pin List</td>
<td>Use the Logical Pin List option to modify the logical pins of the part. This brings up the Logical Pins dialog box.</td>
</tr>
</tbody>
</table>
**View**

The View menu controls the appearance of toolbars and the status bar in the Part Developer UI. The View menu has the following options:

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Bar</td>
<td>Use the Status Bar option to show/hide the status bar in the Part Developer UI.</td>
</tr>
<tr>
<td>Toolbars</td>
<td>Use the Toolbars option to show/hide the toolbars in the Part Developer UI.</td>
</tr>
</tbody>
</table>

**Tools**

The Tools menu provides access to several utilities and checks that ensure that the parts are created correctly. You can also configure Part Developer by using this menu option. The Tools menu has the following options:

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks</td>
<td>Use the Checks option to run the pre-save checks. If any errors are found, they are reported.</td>
</tr>
<tr>
<td>Verify</td>
<td>Use the Verify option to run a number of checks such as view verification, instantiation and packaging, simulation views and against templates.</td>
</tr>
<tr>
<td>ConceptHDL</td>
<td>Use the ConceptHDL option to launch Concept-HDL on the selected symbol.</td>
</tr>
<tr>
<td>PPT Editor</td>
<td>Use the PPT Editor option to launch the Part Table Editor. Part Table Editor is used to create a new part table file or edit an existing one.</td>
</tr>
<tr>
<td>Options</td>
<td>Use the Options option to configure the setup options of Part Developer. You can configure the Properties, Pin, Symbol, Format and Pin Interpretation defaults.</td>
</tr>
</tbody>
</table>
## Templates

The Templates menu provides you the option to create and manage templates for part creation. The Templates menu has the following options:

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Use the New option to create a new part template.</td>
</tr>
<tr>
<td>Open</td>
<td>Use the Open option to open an existing part template.</td>
</tr>
<tr>
<td>Edit</td>
<td>Use the Edit option to edit a part template.</td>
</tr>
<tr>
<td>Extract</td>
<td>Use the Extract option to extract template information from an open part.</td>
</tr>
</tbody>
</table>
Creating Parts

Part Creation Methodology

Part Developer enables you to create and modify symbol, package, part table, and simulation views of parts in Concept-HDL libraries. You should follow the sequence of steps listed below to ensure effective use of the Part Developer tool.

1. Refer to the datasheet for a part.

2. Enter the logical pin information. Logical pins are pin names as they appear on the datasheet.

3. Specify the values in the Setup Options for creating a part. These values are dependent on the standards you follow. These values include properties that are to be placed on the symbols, pins and packages, the placement and load values of the pins. The values also include the format of the symbols as they will appear in Concept-HDL, the format of the low asserted pins, and the interpretation of the pin names. For a more detailed discussion on how to configure the defaults, see Configuring Part Developer on page 26.

4. Create the packages for the part.
   This creates the chips view for the part.

5. Create the symbols for the part.
   This creates the symbol views.

6. If required, create the simulation views. This involves creating the Verilog and VHDL wrappers and map files.

7. Verify the parts that you have created for use in the design flow.

Caution

It is important to follow the methodology as documented above. In case you create the symbols before creating the packages, and save the part, the pin information is lost.
Configuring Part Developer

If you wish to create parts that should contain a set of properties and the display behavior, you can configure Part Developer for generating such parts. For example, you can specify that the pin note size should be 2 grids or 0.10 inch for all the parts that are created or that the input load values should be .01 mA.

Currently, Part Developer supports a pre-defined set of parameters that can be configured. These parameters are as follows:

- Property Defaults
- Pin Defaults
- Symbol Defaults
- Format Defaults
- Pin Name Interpretation
- Save Part Defaults
- Load Part Defaults

**Note**: The defaults you set up for Part Developer are applied only in the current project. If you wish to reuse these defaults for later sessions, you can save them in a .setup file (**File > Save Part Defaults**). The .setup file is written only when you make changes to Part Developer Setup options (**Tools > Options**). If you do not make any changes and choose **File > Save Part Defaults**, Part Developer writes an empty .setup file.

While loading, Part Developer loads the default options from `<install_directory>/share/cdssetup/projmgr/cds.cpm`.

**Setting Up Property Defaults**

You can define the default properties to be attached to every new part you create in the current project. You must specify the views to which a property has to be attached.

To setup the property defaults:

1. Choose **Tools > Options**.

   The *Part Developer Setup Options* dialog box appears.
2. Select the *Property* tab. (This is selected by default.)

3. Enter the property name.

The property name is a user defined variable that you can add to symbols, symbol pins, packages and package pins and art table entries. You can also assign the properties that are used in the PCB front-to-back flow. For more information about such properties, see the PCB Systems Properties Reference.

**Note:** The PACK_TYPE and PART_NAME are reserved properties. You should not add these properties manually. The *Symbol* checkbox gets disabled if you try to add these properties to the symbols. These properties can be added when you run the Symbol Wizard.
4. Enter the value for the property.

For symbol-related properties:

**a.** Specify *Visible* as Name, Value, Both, or Invisible.

The value of the visibility determines what is displayed on the symbol. If you select Name, then only the name of the property will be visible. If you select Value, only the value of the property will be visible. If you select Both, then both the property name and its value will be visible. Invisible hides both the property name and its value.

**b.** Select *Left*, *Right*, or *Center* to specify alignment of properties and values for the symbols.

For example, the part displayed below has the properties PART_LOCATION, PART_OWNER and PART_TYPE with visibility set as Name, Both, and Value:
The part will be displayed in Concept HDL in the following format.

```
Regular
PART_OWNER=XYZ
PART_LOCATION

MYPART
```

5. Indicate the view of the property you have set up. You can specify Symbol (symbol/pin), and Package (package/pin).

6. Click OK to save changes in the project file.

**Setting Up Pin Defaults**

In this *Pin* tab, you can set the defaults for the pins. You can set the position for the different pin types supported by Part Developer and specify the input and output loads.

For example, you can configure the position of the Analog pins to right to ensure that all the Analog pins appear to the right of the symbol.

**Note:** The default values of the pin options for the pins is set according to the Cadence standards. See the Concept-HDL Libraries Reference for standard information on pin options.

To configure the pin defaults:

1. Choose *Tools > Options*. 
The *Part Developer Setup Options* dialog box appears.

2. Select the *Pin* tab.

3. For each pin type, select Position as either *Top, Bottom, Left, or Right*.

4. Enter the Low and High values for the input load. Input Low should be a valid negative load value or *. Input High should be a valid positive load value or *.

5. Enter the Low and High values for the Output load. Output Low should be a valid positive load value or *. Output High should be a valid negative load value.

6. Click *OK* to save changes in the project file.
Setting Up Symbol Defaults

Use the Symbol tab to specify how the symbol should appear in Concept-HDL. This includes values of the units of measurement, the grid size, and the symbol outline.

To setup the symbol defaults:

1. Choose Tools > Options.
   
   The Part Developer Setup Options dialog box appears.

2. Select the Symbol tab.
3. Select the unit of measurement as Inches. The default unit is Inches.

4. Enter the grid spacing. While defining a symbol, Part Developer uses an internal grid spacing that is the same as the grid spacing used by Concept-HDL. Pins are placed on the symbol only at grid points. The default grid spacing is 0.05 inch.

   **Note:** You need to manually synchronize the grid spacing in Part Developer with that of Concept-HDL to ensure that the grid spacing is matched.

5. Select the symbol outline as Thin or Bold.

6. Specify Pin Text Orientation as Automatic, Horizontal, or Vertical. The value of the Pin Text Orientation field determines how the pin notes are displayed in the symbol. The possible values are Automatic, Horizontal, or Vertical. If you choose either Horizontal or Vertical, the text will always display either horizontally or vertically on all four sides of the symbol. If you choose Automatic, all the pin texts that are there on top and bottom will be written vertically and pin texts on the left and right will be written horizontally.

7. Select the *Use Pin Names for Pin Text* check-box if you want to use pin names for pin text. If the check box is selected, the pin name is added as the PIN_TEXT property on the symbol pin.

8. Specify the pin text size. This is in terms of the grid spacing.

9. Specify the minimum symbol height and width.

10. Specify the minimum pin spacing on Left and Right.

11. Specify the minimum pin spacing on Top and Bottom.

12. In case you are creating split parts, then select either the SPLIT_INST and $LOCATION properties or the SPLIT_INST_NAME property to be put on the split symbols. Both these properties ensure that the symbol can be packaged into the same device and get netlisted as a single instance in the simulation netlist. If you use the SPLIT_INST and $LOCATION property, then assign the same value for the $LOCATION property. For example, suppose there is a large pin count device ASYM_PART which is split into four symbols. All the four symbols when instantiated on a design sheet in Concept HDL must have the SPLIT_INST = TRUE property and the same location property value, such as LOCATION =ic1 on it. Similarly, if you use the SPLIT_INST_NAME property and instantiate the part in Concept HDL, then you should ensure that the value of the property is same for all the split symbols, such as SPLIT_INST_NAME = ic1.

13. Click *OK* to save changes in the project file.
For example, displayed below is a part with bold outline and vertical pin text orientation.

![Part Example](image)

**Setting Up Format Defaults**

The Format tab is divided into two group boxes:

- **Low Asserted Pin Syntax.**

  The setting in this group box determines the pin names that Part Developer will treat as low asserted while reading a part. You also specify the notation that Part Developer should use to write a low asserted pin into the symbol and the chips.prt file. You can choose to display the pins with following notations as low asserted pins on the symbol and then write them to the symbol file in one of the following ways:

  - Either with a asterisk (*) or _N suffix: In this case any pin which has either a * or _N in its name is read as a low asserted pin. You can then determine whether you want to write the low asserted pin into the symbol and the chips.prt file with either the '*' or '_N' suffix.

  - With a star (*) notation: In this case, the pins with a * are read as a low asserted pin. The pin is written back to the symbol file with the *.

  - With a _N notation: In this case, the pins with _N are read in a low asserted pin. They are written into the symbol and the chips file with the _N notation.

**Note:** Select the **Use minus [-] sign for low asserted pins in Package view** to ensure that the pins that are considered low asserted are written into the chips file with a minus [-] sign. If you leave this unchecked, and depending on which pins you treat as low asserted, they will get written into the chips.prt file either with a '*' or '_N' notation.

**Note:** You must follow a standard convention for reading and writing the low asserted pins, otherwise the associated pin texts might get lost. For example, while creating a part, you decide on writing the low asserted pins with _N assertion. As a result, all the low asserted pins of the symbol have _N appended to them. Now to modify the part, you load the part back into Part Developer and change the low assertion write back option to *. Now when you save the part, the associated pin text will be lost. This is because when
saving, Part Developer tries to associate pin names with pin texts and fails to find a pin with name, say A*, on the symbol (since the symbol has A_N) and as a result it doesn’t write any PIN_TEXT property for low asserted pins.

- **Chips File Reading Options**

  The options in this group box determines how Part Developer reads the chips.prt file for displaying the following:

  - **Asymmetrical parts**

    Asymmetrical parts have multiple functionality. Corresponding to each functionality, there is a pin list. All sections or slots that have the same functionality have the same pin list. The slots that have the same functionality form one group. Depending on the part, there may be multiple groups. You can choose to display the pin numbers in such a way that the group numbers are also visible. You can do this by selecting the *Use group format for reading asymmetrical parts* check box.

  - **Sizeable pins**

    Sizeable pins can be displayed in Part Developer in one of the following ways: Slot Number-Base Name or Base Name-Slot Number. In the Slot Number-Base Name format, the numeric part is treated as the slot number while the alphanumeric string part is treated as the base name. For example, if the pin name given is 1A, it is taken as base name A and slot number 1.

    In the Base Name-Slot Number format, the alphanumeric string part is treated as a base name and numeric part is treated as a slot number. For example, if the pin name entered is A1, it is treated as a sizeable pin with base name A and slot number 1.

  - **Vector pins**

    Vector pins can be displayed in Part Developer in one of the following ways: Bit Number-Base Name or Base Name-Bit Number. In the Bit Number-Base Name format, the numeric part is treated as the bit number while the alphanumeric string part is treated as the base name. For example, if the pin name given is 1A, it is taken as base name A and bit number 1.

    In the Base Name-Bit Number format, the alphanumeric string part is treated as a base name and the numeric part is treated as a bit. For example, if the pin name entered is A1, it is assigned the base name A and the bit number 1.

To setup the format options:

1. Choose *Tools > Options*.

   The *Part Developer Setup Options* dialog box appears.
2. Select the Format tab.

3. Specify the low asserted pin syntax.

4. Specify whether you want to use the minus sign (-) for low-asserted pins in package view. The default is to use the minus sign.

5. Specify whether or not you want to use the group format to read asymmetrical parts. If this option is checked, you can see the groups to which each of the pin in a package
belongs. For example, displayed below is the expanded pin list of the SOIC package of the LS241 part, where the group format is used to read the asymmetrical part.
The same part is displayed again. This time, the group format has not been used to read the asymmetrical parts (the slot-base name format has been used). Notice that the group information is not displayed in the expanded package pin list.

6. Select the display of the sizeable pins.
7. Choose the display of the vector pins.
8. Click OK to save changes in the project file.

Setting Up Pin Interpretation Defaults

In this section, you can configure the way the pin names will be interpreted. Logical pin names appear in the datasheets in following four patterns:
Alphanumeric String

Usually, scalar pin names, such as A, B, EN and so on appear in data sheets as alphanumeric strings. Any alphanumeric string that is not flanked by numerals falls into the Alphanumeric string category. These pins are always treated as scalar pins.

<Alphanumeric String><Numeral>

Usually, data sheets use the alphanumeric string numeral names for representing vector bits or sizeable pins. For example, the names A0, A1, A2..A7 specify an 8-bit vector bus. Here, A is the base name and 0-7 is the bit range. Similarly, the names B1, B2, B3, B4 specify the sizeable pin B in a part with four slots. The pins can be interpreted in the following ways:

Base Name: The entire alphanumeric string is treated as the base name and it is treated as a scalar pin.

Base Name-Bit: The alphanumeric string part is treated as a base name and the numeric part is treated as a bit number. For example, if the pin name entered is A1, it is assigned the base name A and the bit number 1.

Base Name-Slot: The alphanumeric string part is treated as a base name, and the numeric part is treated as a slot number. For example, if the pin name entered is A1, it is treated as a sizeable pin with the base name A and the slot number 1.

Numeral Alphanumeric String

Usually, data sheets use numeral alphanumeric string names for representing vector bits, sizeable pins, or a scalar pin belonging to a slot group. For example, the names 0A, 1A, 2A..7A specify an 8-bit vector bus or the names 1B, 2B, 3B, 4B specify a sizeable pin B in a part with four slots. Similarly, the scalar pins 1OE and 2OE are used to show that the pin is in two slot groups. The pins will be interpreted in one of the following ways:
Base Name: The entire alphanumeric string is treated as a base name and taken as a scalar pin.

Bit-Base Name: The numeric part is treated as the bit number while the alphanumeric string part is treated as the base name. For example, if the pin name given is 1A, it is taken as the base name A and the bit number 1.

Slot-Base Name: The numeric part is treated as the slot number while the alphanumeric string part is treated as the base name. For example, if the pin name given is 1A, it is taken as the base name A and the slot number 1.

Group-Base Name: The numeric part will be treated as slot group number while alphanumeric string part will be treated as base name. For example, if the pin name given is 1A, it will be taken as the base name A and the slot group number 1.

<numeral>< alphanumeric string>< numeral> notation

Usually, data sheets use the <numeral>< alphanumeric string>< numeral> notation for representing sizeable pins belonging to different slot groups. For example, if a part has 8 slots with slots 1 to 4 forming one group and 5 to 8 forming another group, then a sizeable pin A of that part appears as follows on the data sheet:

1A1, 1A2, 1A3, 1A4
2A1, 2A2, 2A3, 2A4

You can enter these pins in two steps:
- enter 1A1-1A4 and
- enter 2A1-2A4

These pins can be interpreted in the following ways:

Base Name: The entire alphanumeric string is treated as a base name.

Slot-Base Name: The starting numeral is treated as the slot number while the rest of the name is treated as base name. For example, if the name is 1A1, it is taken as pin A1 in the slot as 1.

Bit-Base Name: The starting numeral is treated as bit number while the rest of the name is treated as a base name. For example, if the name is 1A1, it is taken as pin A1 and the bit number 1.
Group-Base Name-Slot: The starting numeral is taken as a slot group number and the end numeral is treated as a slot number. So, if the pin name is 1A1, it is taken as base pin A, slot 1, group 1.

Group-Base Name-Bit: The starting numeral is taken as the slot group number and the end numeral is treated as the bit number. Therefore, if the pin name is 1A1, it is taken as base pin A, bit 1, and group 1.

Base Name-Slot: The end numeral is treated as slot number while rest of the alphanumeric string is taken as the base name. For example, if the pin name is 1A1, it is treated as base pin 1A in slot 1.

Base Name-Bit: The end numeral is treated as bit number while rest of the alphanumeric string is treated as base name. For example, if the pin name is 1A1, it is treated as base pin 1A, and bit 1.

Group-Base Name: The starting numeral is taken as the slot group number while the rest of the name is treated as a base name.

To set up the pin name interpretation:

1. Choose Tools > Options.

   The Part Developer Setup Options dialog box appears.
2. Select the *Pin Name Interpretation* tab.

3. Choose the required pin name interpretation to be followed.

4. Click on *OK*.

**Saving Part Defaults**

The defaults you set up for Part Developer are applied in the current project. If you wish to reuse these defaults for later sessions, you can save them in a `.setup` file. The `.setup` file is written only when you make changes to Part Developer Setup options (Tools > Options).
   The File Browser appears.

2. Specify the location and the setup file name.

3. Click Save.

**Loading Part Defaults**

You can load part defaults by opening the previously saved .setup file. By default, Part Developer loads defaults from <your_install_dir>/share/cdssetup/projmgr/cds.cpm.

   The File Browser appears.

2. Select the .setup file that contains the default information.

3. Click Open.

**Part Creation Situations: Symmetrical, Asymmetrical, and Parts with Pins Split Across Symbols**

While creating parts, you will come across the following situations:
A part has the same the logical pin lists across packages and slots. These are called symmetrical parts. LS00 is an example of a symmetrical part. The pin lists for both DIP and SOIC packages of the LS00 part is displayed below. Note that they are the same.
A part in which the logical pin lists across packages or slots are different. These are called asymmetrical parts, such as LS241.

Different logical pin list across slots: Consider the part LS241. The pins appear in the datasheet as follows:

LS241 is an 8-slot part with a low asserted enable signal (G*) and a high asserted enable signal (G). The high asserted enable signal G is associated with four slots and low asserted enable signal G* is associated with the remaining four. This divides the part into two groups. The first group has G as the enable pin and the second group has G* as the enable pin. Because the functionality of each section is the same, it is also a sizeable
part. However, because of the different assertion signals across the sections, the logical pin list across the section or slots are different. The pin entries created will be as follows:

```plaintext
FILE_TYPE=LIBRARY_PARTS;
TIME='Created/Modified on Wed Apr 05 15:10:24 2000'
primitive 'SN74L3241_SOIC';
  pin
    'A'<0>
      PIN_NUMBER='(2,4,6,8,11,13,15,17)';
      INPUT_LOAD='(-0.01,0.01)';
    'G'
      PIN_NUMBER='(1,1,1,1,0,0,0,0)';
      INPUT_LOAD='(-0.01,0.01)';
    '-G'
      PIN_NUMBER='(0,0,0,0,19,19,19,19)';
      INPUT_LOAD='(-0.01,0.01)';
    'Y'<G>
      PIN_NUMBER='(18,16,14,12,9,7,5,3)';
      OUTPUT_LOAD='(1.0,-1.0)';
  end_pin;
body
  POWER_PINS='(GND;10;VCC;20)';
  CLASS='IC';
  PART_NAME='SN74L3241';
end_body;
end_primitive;
END.
```

Notice that even though the assertion pins G and G* are not present in all the sections, the PIN_NUMBER property values for the pins specify all the sections for the part. The pins that are not present in a given section are specified with a pin number, 0.

**Different logical pin list across packages:** This kind of asymmetry arises due to the need to put power/NC pins on to a symbol. This results in power pins being present in the pin
section for certain packages and on the body section for others. For example, consider the following primitives MYPART_DIP and MYPART_LP:
primitive 'MYPART_DIP';
  pin
    'A'<0>
    PIN_NUMBER='(1,2,3,4)';
    INPUT_LOAD='(-0.01,0.01)';
    'B'<0>
    PIN_NUMBER='(5,6,7,8)';
    OUTPUT_LOAD='(1.0,-1.0)';
  end_pin;
end_body;
end_primitive;
primitive 'MYPART_LP';
  pin
    'A'<0>
    PIN_NUMBER='(1,2,3,4)';
    INPUT_LOAD='(-0.01,0.01)';
    'B'<0>
    PIN_NUMBER='(5,6,7,8)';
    OUTPUT_LOAD='(1.0,-1.0)';
    'VCC'
    PIN_NUMBER='(9*4)';
    PIN_TYPE='POWER';
    'GND'
    PIN_NUMBER='(10*4)';
    PIN_TYPE='POWER';
  end_pin;
end_body;
end_primitive;
Note that for the DIP package, since the power pins are not present on the corresponding symbol, the power pins are present in the body section. However, for the LP package the power pins are on the pin section because they are present in the symbol.

- To create parts where you need to split the pins across the symbol. This is generally done to create symbols for large pin count devices.

Using Part Developer, you can create all the above-mentioned type of parts. While creating a part, you do not have to determine whether the part is symmetrical or asymmetrical. Part Developer can automatically determine the part type and accordingly ask for the required inputs. Examples of creating symmetrical and asymmetrical parts are provided to demonstrate the steps involved in creating parts. For symmetrical parts, see Appendix A. For asymmetrical parts, see Appendix B.

**Creating Parts**

Creating parts involves the following steps:

1. Specifying the logical pin information.
2. Specifying the defaults for the part. See Configuring Part Developer on page 26 for details.
3. Specifying part properties.
4. Creating the packages for the part.
5. Creating symbols for the part.

**Pin Naming**

Part Developer supports the the following as valid pin names:

- alphanumeric characters
- numbers, but only for scalar pins
- #
- $
- %
- +
- =
Part Developer User Guide
Creating Parts

For pins that have ( or ) in their names, hlibftb reports errors if you run verification checks from within Part Developer. However, you can use such pins in Concept HDL schematics by turning off the multi_format_vector option off. Because of this reason, it is suggested that you do not use ( or ) in pin names.

The following are not valid for pin names:

- All extended character sets
- <
- >
- :
- \
- ”
- ,
- ;
- /
- !
- *

Entering Part Properties and Logical Pins Information

To enter part properties for a new part:
1. Choose *File > New > Part*.

   The *New Part Properties* dialog box appears.

   ![New Part Properties Dialog Box](image)

   Enter the following information in the *New Part Properties* dialog box:

2. Select the library name. The default value of this field is the active library with which Part
   Developer was invoked from Library Explorer. A list of libraries in the build area is
   provided as a drop-down list.

3. Specify the part name. This information is used to create the root item of the tree in the
   main Part Developer window. This is the cell name.

4. Specify the physical part name. The default value is the capitalized part name value. If
   part name is changed, this will also be changed accordingly. This entry is used for the
   PART_NAME property in the *chips.prt* file. For technology-independent parts, such
   as LS00, change the physical part name to a complete part name such as SN74LS00.

5. Specify the part type. This can be either IC, IO or DISCRETE. This information goes into
   the *chips.prt* file as the CLASS property.

6. Click *OK* to accept the part properties.
This brings up the *Logical Pins* dialog box.

Enter the following information in the *Logical Pins* dialog box:

1. Click *View* to see the current pin name interpretation.
2. If required, click *Edit* to edit the pin name interpretations.
3. Select the pin type. This combo box has all the pin types supported by Part Developer. The supported pin types are:

   INPUT, BIDIR, TS, TS_BIDIR, OC_BIDIR, OE_BIDIR, OUTPUT, OC, OE, POWER, GROUND and ANALOG

   **Note:** If you choose the POWER pin type, the *Active Low* field gets disabled. You may choose not to specify any pin type by checking the *Unspecified* check box. In this case, the pin type combo-box gets disabled and the added pin shows a type of UNSPEC.

4. Enter the logical pin names. The logical pin names should be entered as they appear on the data sheet. Ranges are supported for logical pins. For example, a pin range like A0-A31, or AA-AZ, or 1A1 – 4A1 can be entered. A comma-separated list of pins and pin ranges like A, B, C, 1A1-1A32 can also be specified.

   **Note:** If in a datasheet, there are multiple instances of a pin with the same name, such as VCC, you need to enter the pin name only once.

5. Click **OK** to accept the logical pin information.

   After you enter the part properties and the logical pins, you need to create the packages for the part.

### Handling of Power and Ground Pins

Power and Ground pins are handled in a special way in Part Developer. If the ground and power pins are in the pin section, then a property called `PINUSE` is added to the `chips.prt` file with the value `POWER` and `GROUND` respectively. This property is required by tools such as `netrev`, Allegro, and SpectraQuest to handle the power and ground pins in the PCB design flow. For example, if a part has a power pin VCC and a ground pin GND and they are both on the symbol, then the `chips.prt` file will have the following entry:

```
'VCC':
    PIN_NUMBER='(2)';
    PINUSE='POWER';
'GND':
    PIN_NUMBER='(3)';
    PINUSE='GROUND';
```

In case the power and the ground pins are on the body section, then the pin numbers go as values of the `POWER_PINS` property. Additionally, a property called `GROUND_NETS` is added to the `chips.prt` file to ensure that the tools in the PCB design flow such as `netrev`, Allegro, and SpectraQuest can handle such pins properly. For example, if a part has a
power pin VCC and a ground pin GND and they are not on the symbol, then the body section of the chips.prt file will have the following entry:

```
body
  POWER_PINS='(VCC:2)';
  POWER_PINS='(GND:3)';
  GROUND_NETS='GND';
  CLASS='IC';
  PART_NAME='POWERONBODY';
end_body;
```

**Creating Packages**

Creating packages for the part involves entering the following information:

- Types of packages, such as DIP, SOIC, and CCC.
- Specifying the footprint information.
- Pin numbers and their mapping with the logical pin names for each package.
- Other package properties, if any.

The information is stored in the chips.prt file, which is created when you create a package.

To create a package:

The *New Package* dialog box appears.

![New Package dialog box](image)

Enter the following information in the General property sheet of the *New Package* dialog box.
1. Enter the physical part name. The default value is taken from the New Part dialog box. The default value can be changed to support technology-independent libraries.

2. Specify whether you wish to enter the package type.

   **Note:** If there is only one package, there is no need to give any type for it. In this case, deselect the “Specify Package Type” check box. This creates one primitive without a type in the chips.prt file.

3. If required, specify the package type.

4. Enter the equivalent package types, if any. This is useful when multiple packages have the same pinouts. By entering them as equivalent packages, you do not need to create separate packages.

5. Specify the reference designator prefix.

6. Determine whether you want to specify the JEDEC_TYPE and ALT_SYMBOLS properties. You can select the JEDEC_TYPE and ALT_SYMBOLS property from the Cadence supplied Allegro footprints.

7. Enter additional properties, if any.

   After you enter the information in the General property sheet, you need to enter the physical pin mapping information.

8. Click the *Physical Pin Mapping* tab.

This displays the Physical Pin Mapping property page. By default, the logical pins are displayed under the Logical Pins column. You need to add the physical pins. You can add the physical pins by either adding them manually, or by extracting the pin numbers from the
footprint information. The *Extract from Footprint* will be enabled only if you add the JEDEC_TYPE value.
Part Developer User Guide
Creating Parts

Package Properties

Logical Pins:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>2A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>3A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>4A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>2B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>3B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>4B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>1Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>2Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>3Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>4Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>POWER</td>
<td></td>
</tr>
<tr>
<td>VCC</td>
<td>POWER</td>
<td></td>
</tr>
</tbody>
</table>

Mapped/Total: 0/14

Physical Pins:

1 unmapped logical and 1 physical pins selected

Specify Physical Pins

- [ ] Extract from Footprint
- [x] Add Manually

Physical Pin Selection: Unmap Unmap All

[ ] Pin Numbers...

[OK] [Cancel] [Help]
1. Click Pin Numbers... to manually add the physical pin numbers that you want to add for the package.

   The *Add Physical Pin Numbers* dialog box appears.

   ![Add Physical Pin Numbers Dialog Box](image)

2. Enter the physical pin numbers in the *Add Physical Pin Numbers* dialog box. The physical pins can either be numeric or alphanumeric as in CGA or BGA. By default, the *Numeric* option is selected. If you want to add alphanumeric pin numbers, choose the *Grid* radio button.

   **Note:** In the Grid option, enter the Row Labels and the Column Labels. The input format of these fields is the same as that for the numeric field except that it can have Alphanumeric ranges also. If labels have characters I or O, they are removed from the range list and a warning is displayed. The pin numbers generated are row major in nature. For example, if the Row Labels is A, B and so on and the Column Labels is 1-5, then the pin numbers are A1, A2, A3, A4, A5, B1, B2, B3, B4 and B5.

3. Click *OK* to accept the physical pin numbers.
Part Developer User Guide
Creating Parts

The Part Developer loads the Physical Pins column in the Physical Pin Mapping property sheet.
4. Choose the physical pin and the logical pin that you want to map it with, and click Map.

   **Note:** If required, you can also sort the Logical Pins list or the Physical Pins list in ascending or descending order. To sort the pins, select the pins that you want to sort and right-click on it. From the shortcut menu, choose the required sorting method.

5. Repeat step 4 for all the physical pins

6. If you have pins that are not connected, select the NC check box.

   **Note:** To map multiple physical pins mapping to a single power pin, for example VCC, choose all the physical pins, choose the logical pin and click Map. Alternately, you can also right-click on the power pin and select the *Replicate* option. You can replicate the power pins the required number of times and then map them with the physical pins.

7. Click OK.

Part Developer creates the package for the part.

**Creating Other Packages**

In most cases, you will have multiple packages for a part. If the part is technology-independent, then packages of several technologies may have the same pinouts, or you may have other packages with different pinouts.

For example, the pinouts for DIP and the CFP package for both SN54LS00 and SN54S00 are same. To save time on creation of packages, Part Developer provides you the facility to make copies of an existing package and change the properties as required.

**Packages With Same Pinouts**

To create other packages which have the same pinouts as an existing package:

1. Select the existing package in the tree view of Part Developer.

2. Choose *Edit > Copy*.

3. Choose *Edit > Paste*.

   This creates a copy of the package with the name `partname_packagename(1)`. For example, if you have a SN5400_DIP package and copy it, then the copy of the package will be SN5400_DIP(1).

4. Select the new package.

5. Right-click on the package, and choose Properties.
The Package Properties dialog box appears.

6. If required, change the physical part name. This will be required for technology independent parts. For example, in both SN54LS00 and SN5400, the package pinout is same. In this case, you need to change the physical part name to SN54LS00.

7. If required, change the package type.

8. If required, add the equivalent packages. For example, the pinout for both DIP and CFP for SN54LS00 is same. Therefore, you should add CFP as an equivalent package.

9. Click OK to complete the creation of the package.

10. Repeat steps 1-9 for each package.

Packages with Different Pinouts

When you create other packages where the pinouts are different from an existing package, such as the CFP package of the SN5400 part, you may have the following situations:

- If the pinouts are very different from the existing package, it is easier to create the new package. The process to create a new package is described in the Creating Packages section in Appendix A.

- If the pinouts are similar, you can create the other packages by modifying the pin mapping information.

To modify the pin mapping information:

1. Select the existing package in the tree view of Part Developer. For example, select the SN5400_DIP package.

2. Choose Edit > Copy.

3. Choose Edit > Paste.

4. This creates a copy of the package with the name partname_packagename(1), for example, SN5400_DIP(1).

5. Select the new package.

6. Right-click on the package, and choose Properties from the shortcut menu.

The Package Properties dialog box appears.

7. If required, change the physical part name. This will be required for technology-independent parts. For example, in both SN54LS00 and SN5400, the package pinout is same. In this case, you need to change the physical part name to SN54LS00.
8. Right-click on the selected package.


10. Choose the Physical Pin Mapping tab.

11. Press Ctrl + Click to select the pins for which the pin mappings are different.

12. Choose Unmap to unmap the pin mappings.

13. Remap the pins to the required logical pins.

14. Click OK to complete the creation of the package.

**Selecting a Default Package**

Using Part Developer, you can select a package as the default package. To select a default package:

- Select the check box next to the package name in the Packages tree view.

**Creating Symbols**

To enable you to create symbols effectively, Part Developer provides a Symbol Creation Wizard. The Wizard guides you through the symbol creation process.

- For symmetrical parts, see Appendix A.
- For asymmetrical parts, see Appendix B.
- To create a symbol with Power/NC pins, see Appendix A.
- To create parts with pins split across symbols, see Appendix D.

**Split Part Methodology**

When you use split symbols in a schematic, you need to ensure they are packaged into the same device and get netlisted as a single instance in the simulation netlist. For this one of the following needs to be done:

- Add the SPLIT_INST_NAME property on each of the split symbol, or
- Add the the following two properties are required on each of the split symbols:
  - SPLIT_INST=TRUE
$LOCATION = ?

When you instantiate these split symbols in Concept-HDL, assign the same value for the $LOCATION property.

For example, suppose there is a large pin count device ASYM_PART which is split into four symbols. All the four split symbols when instantiated on a design in Concept-HDL must have the SPLIT_INST=TRUE property and same location property value, for example LOCATION=ic1 on it.

When creating a split part in Part Developer, you need to select either the SPLIT_INST_NAME property or the SPLIT_INST and $LOCATION property from the Tools > Options > Symbol dialog box. The selected property is added automatically by the tool.

Creating Flat Symbols

Using Part Developer, you can create flat symbols of sizeable parts. To create flat symbols:

1. Create the symbol.
2. Right-click on the symbol.
3. Choose the Fixed Size option.
   The Fixed Size Symbol dialog box appears.
4. Enter the value for the Size field.
5. Click OK.
Modifying Parts

Overview

Using Part Developer, you can modify existing parts. You can do the following to modify the parts:

- Modify the logical pin list
- Modify the packages
- Modify the symbols
- Modify the Verilog wrappers/map files
- Modify the VHDL wrappers/map files

Modifying the Logical Pin List

After you have created a part, you may want to add, remove, or rename one or more logical pins. Using Part Developer, you can easily add or remove logical pins.

Removing Logical Pins

To remove logical pins:

1. Expand the Logical Pins entry in the Part Developer window.
2. Right-click on the pin that you want to delete.
3. Choose Delete.

   This deletes the selected pin from the logical pins list. The packages and the symbols are also updated automatically.
Part Developer User Guide
Modifying Parts

Adding Logical Pins

To add logical pins:

1. Right-click on the Logical Pins entry in the Part Developer window.
2. Choose Edit Pin List.
   
   The Logical Pins dialog box appears.
3. Add the logical pins.
4. Click OK.
   
   The logical pins are added to the logical pins list.

Note: After you have added the logical pins, you need to update the packages and the symbols (in that order) to add the logical pins to them.

Renaming Logical Pins

Using Part Developer, you can rename logical pins. To rename a logical pin:

1. Expand the Logical Pins entry in the Part Developer window.
2. Right-click on the logical pin that you want to rename.
3. Choose Rename.
   
   The Rename Logical Pin dialog box displays.
4. Enter the new pin name.
5. Click OK.
   
   This updates the pin name in the logical pin list. The pin name in the packages and symbols are also changed automatically.

Note: If the pin name is also used as the pin text, changing the pin name will not automatically update the pin text. If required, you will need to change the pin text manually.

Modifying Packages

Using Part Developer you can modify existing packages. To modify a package:

- Change the properties.
Part Developer User Guide

Modifying Parts

- Modify the physical pin mapping.
- Modify the footprint.

**Note:** When you modify a package, you should modify the associated symbols too. To modify the symbols, right-click on the symbol and choose *Modify Pin List.*

### Changing Package Properties

Using Part Developer, you can change the properties of a package. To change the properties:

1. Right-click on the package that you want to modify.
2. Choose *Properties.*
   
   The **General** tab of the **Package Properties** dialog box appears.
3. Change the required properties.
4. Click *OK.*

**Note:** If you change the Physical Part Name or Package Type property, the PART_NAME and the PACK_TYPE properties in the symbols that are associated with the package are also updated.

### Modifying Physical Pin Mappings

The Physical Pin Mappings can be changed for a package. You can add and remove physical pins and alter the existing pin mappings.

To add physical pins:

1. To add more physical pins, click *Add Numbers.*
   
   The **Add Physical Pin Numbers** dialog box appears.
2. Enter the pin numbers.
3. Click *OK.*

   The added physical pins will appear in the Physical Pins list.

To remove physical pins:

1. Unmap the physical pin to be deleted.
2. Right-click on the unmapped pin in the Physical Pins list.
3. Choose *Delete* to delete the pin.

4. Click *OK* to accept the changes.

To altering existing pin mappings:

1. Unmap the pins whose mappings are to be modified.
2. Remap the pins as per the new requirements.

### Modifying Symbols

Symbols can be modified to:

- Change the properties such as Part Name and symbol pin types.
- Change the associated package type.

**Note:** When you modify a package, you should modify the associated symbols too. To modify the symbols, right-click on the symbol and choose *Modify Pin List*.

- Modify the symbol pin list when logical pins are added or removed.
- Change the order of pins in a symbol.
- Change the position of pins in a symbol.

### Modifying Symbol Properties

To modify symbol properties:

1. Right-click on the symbol to be modified.

2. Choose *Properties*.
   
   The *Symbol Properties* dialog box appears.

3. Modify the symbol properties.

4. Click *OK*.

### Changing the Associated Package Type

To change the package type associated with a symbol:

1. Right-click on the symbol that you want to modify.
2. Select *Modify Pin List*.
   The Symbol Modification Wizard page appears.

3. Select *Yes*.
4. Click *Next*. 
The Symbol Modification Wizard page appears.

5. Select the packages that you want to associate with the symbol.

6. Click Next.

   The Summary page appears.

7. Click Finish to complete the symbol modification process.

**Adding a Logical Pin to All Symbols**

To add the logical pin to all the symbols:

1. Right-click on the Symbols tree item.

2. Choose Add Logical Pins.
3. The Add Logical Pins dialog box appears. This dialog box displays all the new logical pins that can be added to the symbols.

4. Select the logical pins that you want to add to all the symbols.

5. Click OK.

This adds the logical pins to all the symbols.

**Note:** After you update the symbols, you should also update the associated packages to ensure that the symbols are in synchronization with the packages.

### Adding Logical Pins to Specific Symbols

Updating the symbol with the new logical pins is a two-step process. First, you need to add the pins to the package associated with the symbol. Once the package has been updated, you need to update the symbol. To update the symbol:

1. Right-click on the symbol you want to update.

2. Choose *Modify Pin List*.

   This displays the *Symbol Modification Wizard*.

3. Follow the steps in the Wizard to complete the process of modifying your symbol.

### Changing Pin Orders in a Symbol

Using Part Developer, you can change the order of pins as they appear in a symbol. For example, for a part, a symbol might have the pins A, B and C appearing from top to bottom. Using Part Developer, you can change the order to A, C and B. To change pin order:

1. Right-click on the symbol whose pin order you want to change.

2. Choose *Properties*.

   The *Symbol Properties* dialog box appears.

3. Choose *Symbol Pins* tab.

4. Change the order of the pins by selecting the pin and clicking on either the Up arrow or the down arrow.

5. Click *OK* to accept the changes.

**Note:** You need to regenerate the symbol to ensure that the changes made in the pin ordering is reflected in the symbol.
Changing Pin Positions in a Symbol

Using Part Developer, you can change the position of pins as they appear in a symbol. For example, for a part, a symbol might have an input pin A appearing in the left of the symbol. Using Part Developer, you can change the position of the input pin from left to right. To change pin position:

1. Right-click on the symbol whose pin position you want to change.
2. Choose Properties.
   The Symbol Properties dialog box appears.
4. Change the position of the pins by selecting the pin and selecting the required pin position from the Position drop-down list box.
5. Click OK to accept the changes.

Note: You need to regenerate the symbol to ensure that the changes made in the pin ordering is reflected in the symbol.

Modifying Split Symbols

When you modify split symbols, the you need to be aware of the following:

1. Creating new symbols.
   To create additional symbols, you need to resplit the pins across a larger number of symbols. For example, if you have two symbols for a part and want to create a third one, you will need to specify that you need three symbols and resplit the pins across them. Part Developer provides you with a Symbol Modification Wizard to enable you to resplit the pins across the symbols.

2. Copying symbols.
   You may want to make copies of a symbol to create a flat symbol from a sizeable symbol. However, for parts with split symbols, you cannot simultaneously have both sizeable and flat symbols. You can have either a symbol with all the vector pins/sizeable pins expanded or a symbol with the vector/sizeable pins contracted.

3. Deleting Symbols
   When deleting split symbols, one of the following is possible:
   - Deleting all the symbols.
Decrease the number of symbols by resplitting the pins across a smaller number of symbols. A Symbol Modification Wizard helps you in effortlessly decreasing the number of symbols and resplitting the pins across the smaller number of symbols.

4. Adding Logical Pins

If you add pins to the logical pins list, you cannot globally add them to all the symbols. You need to run the Symbol Modification Wizard to add the logical pins to the required symbol.

**Note:** You should map the added logical pins to physical pin numbers in the package before you run the Symbol Modification Wizard.

5. Modifying/ Deleting Logical Pins

If you add or delete a logical pin and then try to save the part, it will ask you whether you want to regenerate the symbols on which the pins have been added or from where the pins have been deleted. If you choose to regenerate the symbols, they will lose their existing shape and get saved as rectangular symbols.

To modify the split symbols:

**a.** Right-click on a symbol.

**b.** Select *Modify Pin List.*
Handling Pin Texts

Pin Text Methodology

In parts created in earlier versions of Part Developer, the pin text was stored as a note. Therefore, there was no direct association of the pin text with the pin. In the new release of Part Developer, pin texts are stored properties on the pin.

In the new Part Developer, the following methodology is implemented:

- For new parts, the pin text is stored as the PIN_TEXT property. This property is associated with the pin.

- When parts created in an earlier version of Part Developer or Concept HDL are loaded in Part Developer, it is likely that the pin texts on symbols are not stored as properties on the pin. In this case, Part Developer will automatically assign the pin texts to the pins and you will be prompted to save the part. However, before saving the part, you should verify that the pin texts are mapped correctly by launching the Associate Pin Text dialog box and checking the pin texts corresponding to the pins.

- In case there are some texts that are still unassociated, you should decide whether you want to retain or remove the text. In case there are any unassociated pin text left on the symbol, a warning will be given during the saving of the part. You can choose to either retain the unassociated text in the symbol or remove them from the symbols.

The next time you load such parts, the pin texts will be associated with the pins as pin properties.
Overview

**Note:** This feature is only available in the Part Developer Expert version. This version is available with the PCB Librarian Expert licence.

Concept-HDL generates a netlist that you can use to simulate your logical design using a Verilog or VHDL based simulator. To generate the netlist, it uses the symbol pin names as the interface ports for each component used in the design. For each component, the symbol pin names are saved as ports in a VHDL entity declaration file in the entity view of the part. The entity declaration from the symbols may be different from the actual VHDL model's entity declaration. Therefore, to successfully simulate a design, it is required that the port names
generated for a component in the HDL netlist match the port names in the actual VHDL model. For example, for the part 74ac74, the port names in the entity view are as follows:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use work.all;
entity \74ac74\ is
  generic (
    size:positive:= 1
  );
  port (
    \cl\: IN STD_LOGIC;
    CLOCK: IN STD_LOGIC;
    D: IN STD_LOGIC_VECTOR (size-1 DOWNTO 0);
    \pr\: IN STD_LOGIC;
    Q: OUT STD_LOGIC_VECTOR (size-1 DOWNTO 0);
    \q\: OUT STD_LOGIC_VECTOR (size-1 DOWNTO 0);
  end \74ac74\;
```
For the same part, the VHDL model's port names or entity are:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use ieee.std_logic_arith.all;
use std.textio.all;
use std.textio;

entity sn74ac74 is
  generic (
    -- fclk      : time := 7.2 ns; -- max clock frequency = 140 Mhz
    tw_cntl    : time := 5 ns; -- min pulse duration for pre* and clr*
    tw_clk     : time := 5 ns; -- min pulse duration for clk
    tsu_cntl   : time := 0 ns; -- max setup time pre* and clr*
    ts_data    : time := 3 ns; -- max setup time for data
    th_data    : time := 0.5 ns; -- max hold time for data
    tp_cntl_max : time := 10.5 ns; -- max prop delay for cntl to Q
    tp_clk_max : time := 10.5 ns -- max prop delay for data to Q
  );

  port (  
    pre1_n : in std_logic := 'U';
    clr1_n : in std_logic := 'U';
    d1     : in std_logic := 'U';
    clk1   : in std_logic := 'U';
    q1     : out std_logic;
    qb1    : out std_logic;
    pre2_n : in std_logic := 'U';
    clr2_n : in std_logic := 'U';
    d2     : in std_logic := 'U';
    clk2   : in std_logic := 'U';
    q2     : out std_logic;
    qb2    : out std_logic
  );

end sn74ac74;
```
So the mechanisms that are employed to map the symbol pin names (or port names in the entity view) with the port names in the VHDL model are files called wrappers and map files. The wrappers or map files can be created using Part Developer.

Creating a VHDL Wrapper/Map File

VHDL wrappers or map files can be created using Part Developer. When creating a VHDL wrapper or map file, specify the following information:

- The path to the VHDL model. This can be done by specifying either the actual physical path to the VHDL model or the Lib:Cell:View structure.

  For example, to access the VHDL model stored in x:⁄74ac⁄sn74ac74, you may provide either the entire physical path or use the lib:cell:view method, i.e. specify sn74ac:sn74ac74::vhdl_lib.

  **Note:** To use the lib:cell:view method, there should be a library entry in the cds.lib file of the project. For example, to use the sn74ac74 with the lib:cell:view method, the following entry should be present in the cds.lib file:

  ```
  DEFINE 74ac x:⁄74ac
  ```

- Determine whether to bind the VHDL model and the symbol. That is, determine whether to bind the symbol with one specific architecture (behavior). If you decide to bind the VHDL model and the symbol, then the binding statement goes into the wrapper. This is an optional step.

  **Note:** Since each view of the model is essentially a specific architecture, selecting the VHDL model by using the lib:cell:view method automatically fills in the binding statement. If you specify the actual physical path to the VHDL model, then you have to explicitly enter the binding information. If you decide not to enter the binding information during wrapper creation, then the binding information has to be specified later in the simulation flow. However, not providing the binding information in the wrapper provides the freedom to use the same wrapper for different architectures.

- Enter the mappings between the symbol pins and the model ports.

- Determine the generics, and its type and value and whether they need to be annotated on the symbol.

- Determine whether to create only a wrapper or a map file or both.

The sn74ac74 part is used to demonstrate the steps in creating a VHDL wrapper. The VHDL model for the part is located in c:\demo\test\small_medium_ics_32

1. Choose *File > New > VHDL Wrapper/Map File.*
The VHDL Wrapper/Map file dialog box appears.

![VHDL Wrapper/Map File Dialog Box](image)

Enter the path to the VHDL model. This can be done either by specifying the lib:cell:view path or by physically selecting the model file. The File method is used to specify the path to the VHDL model.

2. Select *File*.

3. Browse to the location of the VHDL file, `c:\demo\test\small_medium_ics\sn74ac74\entity\vhdl.vhd`.

4. Click *Port Mapping*. 
The **VHDL Symbol Pin to Model Port Mapping** dialog box appears.

5. Select the pin CL* in the Pin List column.


7. Click **Map**.

   **Note:** In case you are mapping a symbol pin to a model port of different modes, then the Update pin mode dialog box appears. Similarly, if the types are different, then the Update pin type dialog box appears. In this dialog box you can determine whether or not to map the pins of different types. To suppress this dialog box, you can click the **Automatically Update pin mode** option.

8. Click **OK**.

9. Repeat steps 5 to 8 for all the pins that are to be mapped.

10. Enter the Lib Cell View entries and select the Bind the model and symbol if the wrapper is to be bound to a specific architecture.
11. Click OK.

Next, determine the generics that should be annotated to the symbols.

12. Click Generics.

The Select VHDL Generics dialog box appears.

13. Determine the generics that you want to annotate to the symbol.

14. Click OK.

15. To simultaneously create a map file, click Setup.

16. Select the Generate Map File option if you want to create the map file.

17. Click OK.

18. Click OK.

This completes the creation of a VHDL wrapper/map file.
Modifying a VHDL Wrapper/Map File

Due to certain reasons, such as to change the binding information or generic informations, you may need to modify the existing VHDL wrappers. Using Part Developer you can modify existing VHDL wrappers. The steps are:

1. Right-click on the VHDL wrapper which is to be modified.
2. Select Properties.
   
   The VHDL Wrapper/Map File dialog box displays.
3. Change the values as required.
4. Click OK.

Deleting a VHDL Wrapper/Map File

To delete a VHDL wrapper:

1. Select the VHDL wrapper to be deleted.
2. Press Delete.
3. Click Yes.

Renaming a VHDL Wrapper/Map File

To rename a VHDL wrapper:

1. Right-click on the VHDL wrapper to be renamed.
2. Select Rename.
3. Enter the name for the wrapper.
4. Click OK.
Creating Verilog Wrappers/Map Files

Overview

**Note:** This feature is only available in the Part Developer Expert version. This version is available with the PCB Librarian Expert licence.

To ensure that Verilog XL can simulate the parts created using Part Developer, a mapping between the pin names of the part with the corresponding ports in the Verilog model is required. This mapping information is stored in a Verilog file termed Verilog wrapper or map file. The pin names in the wrappers are read from the entity view and the port names from the Verilog model.

**Note:** The entity view gets created automatically when the symbol view is created.

You can create the wrappers using Part Developer.

The LS00 part will be used to detail the steps to work with Verilog wrappers.

**Creating a Verilog Wrapper and Map File**

1. Choose *File>New>Verilog Wrapper.*

   The *Verilog Wrappers/Map File* dialog box appears.

   **Note:** The Verilog model file can be accessed by specifying either the physical path to
the model file or the lib:cell:view structure. To access the model file by using the lib:cell:view method, ensure that the library definition exists in the cds.lib file.

2. Click browse to choose the Verilog model. For LS00, select the SN74LS00.v file from the CDS_INST_DIR/sun4v/veriloglib/verilogTTL/74LSTTL directory.

3. Click OK.

You will notice that under the pin list entry, there are only three pins a, b, and y, whereas under Model Ports there are fourteen pins. This is because the part that you have created is a sizeable part, and therefore pin names of only one section is visible, whereas the Verilog model is a flat model and therefore all ports are displayed.

4. Select y from the pin list.

5. Select _1Y from the Model Ports list.

6. Click Map to map the pin to the port.

   Note: In case you are mapping a symbol pin to a model port of different modes, then the Update pin mode dialog box appears. Similarly, if the types are different, then the Update pin type dialog box appears. In this dialog box you can determine whether or not to map the pins of different types. To suppress this dialog box, you can click the Automatically Update pin mode option.

7. Similarly map a with _1A and b with _1B.

8. Click OK.

   Next, determine the parameters to be annotated to the symbol.

9. Click Parameters.
The \textit{Select Verilog Parameters} dialog box appears.

10. Select the \textit{Annotate on Symbols} check box for the parameters that are to be annotated to the symbols.

11. Click \textit{OK}.

12. To simultaneously create a Verilog map file, click \textit{Setup}.

   The \textit{Setup for Verilog Wrapper/Map File} dialog box appears.
Part Developer User Guide  
Creating Verilog Wrappers/Map Files

13. Select Generate Map File to create a Verilog map file.

14. Click OK.

15. Click OK.

   This creates the Verilog wrapper for the LS00 part.

Modifying a Verilog Wrapper/Map File

After creating wrappers/map files, you may decide to modify the information contained in the wrapper, such as changing the Verilog model or the mapping information etc. Existing wrappers can be modified using Part Developer. To modify a Verilog wrapper:

1. Right-click on the wrapper.

2. Select Properties.

3. Make the required changes, such as changing the Verilog model or mappings and so on.

4. Click OK.

Deleting a Verilog Wrapper/Map File

To delete a Verilog wrapper:

1. Select the Verilog wrapper.

2. Press Delete.

3. Click Yes to confirm the deletion.

Renaming a Verilog Wrapper/Map File

When more than one Verilog wrappers/map file is created, the new wrappers are named vlog_model[n]. You may want to change the name to a more intuitive one. Part Developer allows you to rename the Verilog wrappers.

To rename a Verilog wrapper:

1. Right-click on the wrapper to be renamed.

2. Select Rename.

3. Modify the wrapper name
4. Click OK.
Part Templates

Overview

Note: This feature is only available in the Part Developer Expert version. This version is available with the PCB Librarian Expert licence.

Part Developer provides you the ability to set up the part construction rules and property setup through the use of Part Templates. Then, using the templates, you can quickly create parts that follow your company standards and yet have the flexibility to add properties and behaviors specific to the part.

Using Part Developer, you can:

1. Create a template.
   You can create a template with data such as property names and values, pin positions and symbol display characteristics. The template is then used to create multiple parts.

2. Create parts using a template.
   You can select a template from the existing list of available templates and create parts based on the information stored in the template.

3. Verify parts against a template.
   You can verify parts against a selected template or the original template from which the part was created.

Creating a Part Template

Part Developer enables you to create part templates. While creating part templates, you can specify the following information:
Packages
- Package properties
- Pin load properties

Symbols
- Grid size
- Symbol outline
- Pin text orientation
- Pin text size
- Whether to use pin names for pin text
- Minimum symbol height
- Minimum symbol width
- Symbol properties
- Symbol pin spacing, both top/bottom and left/right
- Position of symbol pin types

After you specify the information, you save the information in a .tpl file. This file can then be used as a template in later sessions of Part Developer to create parts.

To demonstrate the steps in creating a part template, a template file with the symbol property of PART_NAME and units of measurement in metric will be created and saved as mytemplate.pl.

The steps are as follows:

The New Template dialog box appears.

2. Enter the property name as PART_NAME.
3. Enter the property value as ?.
   This will ensure that the property is annotated to the symbol and the value for this property has to be specified by you in Concept-HDL.
4. Select the value of the Visible field as Both.
5. Specify the alignment as Center.
6. Click the Symbol tab.
7. Select the value of Units field as Metric.
8. Click Save.
9. Enter the file name as mytemplate.
10. Click Save.

This completes the creation of a new part template. After you complete the creation of the template, the tool will ask you whether you want to apply the template in the current
session. Choose Yes if you want to create parts based on the template in the current session of Part Developer; otherwise, choose No.

Using Part Template to Create Parts

You use a part template as the building block in creating one or more parts. To create parts using a part template, you need to open the part template and apply the template values to the current session of Part Developer. This overrides the default values specified through Tools>Options. Once you have applied the template, proceed with the creation of parts.

The steps to use a template to create parts is as follows:

1. Select Templates>Open.
2. Select the template that you want to load.
3. Click Open.
4. Click Apply.

This applies the value stored in the part template to the current session of Part Developer.

In case you already have a part open then the loaded part will be updated for the following:

- Pin Loads based on pin types.
- Package Properties, except for special properties that are not allowed in the Properties section such as PART_NAME.

To apply the template related changes in symbols, you will need to regenerate them.

Verifying a Part Against a Template

To ensure that the parts are conforming to your standards, Part Developer enables you to verify a part against a template. The verification is done only for those values that exist in the template file. The output is displayed as a report that can be saved as a .rep file.

To demonstrate the steps to verify a part against a template, the ls00 part and the myplate.tpl will be used. The steps are as follows:

1. Open the part.
2. Choose Tools>Verify.
3. Select the *Verify with Template* radio button.

4. Click *OK*.

5. Browse and select the template against which you want to verify the part.

6. Choose *Open*.

   The verification test report displays.

---

### Extracting Templates from Existing Parts

If you have a part which has been built in compliance with your company standards, you can extract information from it and create a template. After the template information is extracted, the part is verified against the extracted template information. A verification report is generated listing the differences with the extracted values. Template information is extracted as per the following rules:

- **From Packages**
- **Pin Load Extraction**
- **Symbol Data Extraction**
- **Symbol Property Extraction**
- **Symbol Pin Extraction**
- **Grid Extraction**
- **Minimum Size Extraction**

#### From Packages

- All properties found in any of the packages are added to the template with their values.
- If the property name matches with any of the properties listed below then the value is replaced with "?":
  - JEDEC_TYPE
  - POWER_PINS
  - NC_PINS
  - BODY_NAME
  - PART_NAME
Pin Load Extraction

Pin Load is extracted from the different pin types. If a pin type is not found on one of the packages, it is searched for in the next package and so on. If a pin type is not found on any of the packages, its load is not added to the template. If for any pin type the load extracted is not a standard, i.e., the load is not same for all pins of a type on all packages then the first load is picked up.

Symbol Data Extraction

- All symbols are read for a given part.
- All symbols must have at least one connection with a line stub or a bubble else an error stating that process cannot proceed is displayed.

Symbol Property Extraction

- All properties found in any of the symbol are added to the template with their values.
- If a property exists both in a package and a symbol, the package property is given precedence and its value is extracted.
- Alignment and visibility are extracted from the symbol and added.
- If the property differs in value across packages or across symbols to or it differs in visibility or alignment, then the value of the property from the last instance of either the package or symbol is extracted.

Symbol Pin Extraction

For each pin the following checks are done:

- A search is done for the first pin of each pin type and its location value is added to the template.
- The _Use Pin Names For Text_ option is set to false if a single pin is found to violate this rule.
- The Text Style for Pin Notes is interpreted. If the style is vertical for top and bottom pins and horizontal for pins on left and right then the style is considered to be Automatic. The angles of 90 and 270 are considered equivalent and vertical and 0 and 180 are
considered equivalent and horizontal. If it is not consistent for all the instances for a pin type based location then user is warned that this is not a standard.

Grid Extraction

The highest common factor of all differences in X distances and Y distances is taken as the minimum grid unit. It is calculated into template in Inches units.

Minimum Size Extraction

- The minimum pin spacing values on the left and right are read for all the symbols and the smallest value is extracted to the templates as the minimum spacing value for left and right. If in a symbol, 0 or 1 pin exist on left and right, its value is not extracted.

- The minimum pin spacing values on the top and bottom are read for all the symbols and the smallest value is extracted to the templates as the of minimum spacing on top and bottom. If in a symbol, 0 or 1 pin exist on top and bottom, its value is not extracted.

- The symbol height value is read for all the symbols and the smallest value is extracted to the templates as the of minimum symbol height.

- The symbol width value is read for all the symbols and the smallest value is extracted to the templates as the of minimum symbol width.

- The Outline is extracted as thick if all outlines are thick. The Outline is extracted as thin if all outlines are thin. If neither of the cases are true then no extraction is done for this value. If a value is not extracted, a warning is displayed.

To extract the templates:

1. Load the part.
2. Choose Templates>Extract.
3. Specify the location and the name for the template file.
4. Choose Save.

This creates a template with data extracted from the loaded part.
Using XML

Overview

Note: This feature is only available in the Part Developer Expert version. This version is available with the PCB Librarian Expert licence.

The growing popularity of the Internet, along with the possibilities that XML provides has resulted in companies providing part information in XML format. Part Developer enables you to create parts by importing part information available in XML format. In this release of Part Developer, the E-tools XML format is supported. The E-tools XML datasheets are provided by the Electron Tools Company. For more information, see www.e-tools.com.

Creating Parts Using the E-tools XML Format

Part Developer enables you to create parts from the E-tools XML documents.

Note: The current setup values will get applied to the imported part.

The steps to create a part from an XML datasheet are:

1. In Windows NT, add the following to your path settings:
   <your_install_dir>/tools/jre/bin

2. Choose File > Import > XML.

   The Import from XML To Concept HDL dialog box appears.

**Note:** The XML file can be either in the .xml or .zip format.

4. Decide on whether you want to convert only the master component, or each alias as individual primitive or each alias as individual cell.

   If you choose to convert only the master component, then one Concept HDL part is created. The chips.prt file will have only one primitive section with the master component name as its only entry.

   If you choose to convert each alias as individual primitives, then one Concept HDL part is created with each alias appearing as a primitive entry in the chips.prt file. For example, if a part has four aliases, the resultant chips.prt file will have four primitives.

   If you choose to convert each alias as an individual cell, then as many number of parts as there are aliases will be created. For example, if a part has four aliases, then four separate Concept HDL parts will be created.

5. Select the Concept HDL library where you want to store the part.

6. Click *OK*.

   The part displays in the Part Developer window.
Post Import Issues

After you read-in the part information from the XML datasheet, you need to take care of the following issues:

- PIN_TYPE Property
- CLASS Property
- Duplicate Pins
- Pins With ‘>’ or ‘<’
- Partnames with ‘/’
- JEDEC_TYPE
- NC Pins
- Symbol Interpretation

PIN_TYPE Property

The PIN_TYPE property gets filled in as per the following rules:

- Passive is interpreted as ANALOG.
- Rest of the pins are mapped to their corresponding types in Part Developer.

CLASS Property

The CLASS property gets filled in as per the following rules:

- CLASS property gets the value IC if the Designator element in the XML datasheet had the value U.
- CLASS property gets the value IO if the Designator element in the XML datasheet had the value J, JP or SW.
- CLASS property gets the value DISCRETE for all other values of the Designator element type. These values are: C, D, ISO, L, LS, Q, R, T, U, VR, and Y.

Therefore, you may need to check the value of the CLASS property after the data has been imported from the XML datasheet. If required, change the value of the CLASS property. The steps are:
1. Right-click on the part name in the tree view.
2. Select *Properties* from the shortcut menu.
3. Change the value of the *Part_Type* field.

**Duplicate Pins**

Often a part may have duplicate pins names such as multiple collector pins in transistor, programmable IO pins in FPGAs etc. In such cases, Part Developer appends an index with ‘_’ to the pin name starting from the second occurrence. For example, if there are two collectors with name C, they will be read-in as C and C_1.

**Pins With ‘>’ or ‘<’**

Certain pins have either a ‘>’ or a ‘<’ symbol in their pin names, such as P>Q or P<Q in comparators. Such pins will be read in with the ‘>’ symbol converted to ‘_GT_’ and the ‘<’ symbol converted to ‘_LT_’ in the pin names. For example, a pin name P>Q will get converted to P_GT_Q.

**Partnames with ‘/’**

Some part may have ‘/’ symbol in their part names. The ‘/’ in such part names will get converted to ‘_’ in the primitive entry of the *chips.prt* file.

**JEDEC_TYPE**

When a part information is imported from an XML datasheet, the JEDEC_TYPE entry is not seeded. You will need to manually specify the value of the JEDEC_TYPE property for a part.

**NC Pins**

E-tools XML datasheets do not contain information about the NC pins. Therefore, when you import the data from the XML datasheets, NC pins would be absent from the packages. You can follow one of the two methods to add the NC pins to the packages:

1. Use the *Extract From Footprint* option in the Physical Pin Mapping tab of the Package Properties dialog box, and then mark the required unmapped pins as NC pins.
2. Manually add the pin numbers and mark them as the NC pins.
Note: If you use the Extract From Footprints option, you will lose the mappings for those pins that are mapped to logical pins but not present in the list of extracted pin numbers.

Symbol Interpretation

After you create a part from an XML datasheet, you need to be aware of the following:

- The ORIGIN of the symbol will be at the top left corner of the symbol bounding box and the PART_NAME will be displayed just above the ORIGIN.
- Pin text will appear outside the symbol graphic.
- In data sheets, the multi-assertion pin names appears as name/\name. When such pins are read into Part Developer from XML, each character in the low asserted part of the name is followed by a '\'. For example, RD/W\R will be written as RD/W\R\.
- Sometimes, some of the symbols may appear to have additional height. This is because when a part data is imported from XML, all the power pins are moved to the body section of the chips.prt file. This results in the removal of power pins from the symbols. However, the space reservoir by the symbol for power pins is not adjusted after the power pins are moved to the body section.

Saving Part Information in XML Format

Using Part Developer you can save the part information in XML format. Essentially, the symbol and chips view of Concept HDL part is saved in the XML format. The XML format exported out is in the schema of the E-Tools DTD. For more information on the DTD refer to www.e-tools.com.

Note: Concept HDL supports multiple packages through the primitives in the chips.prt file. Each of these primitives map to a unique XML Part. The user needs to select the primitive that needs to be saved in the XML format.

The translation supports

- Normal flat-parts.
- Sized parts. These are converted to single cells in XML
- Split parts. These are converted to multiple cells in XML having the same source package.
- Asymmetrical parts. These are converted to multiple cells in XML having the same source package.
Steps

The steps to save a part information in XML format are:

1. Launch Part Developer.
2. Load the part that you want to save in the XML format.
3. Select File > Export > XML.
   
   The Export from Concept HDL to XML dialog box appears.
4. Select the Concept HDL library and the cell whose data you want to save in the XML format.
5. Click OK.
   
   The Export from Concept HDL to XML - Select Package dialog box appears.
6. Select the package whose information is to be saved in the XML format.

Note: In this release, Part Developer makes separate XML files for each of the packages of a part.
7. Select the directory where you want to store the XML file and click OK.
   
   The XML file name is derived from the package name of the Concept HDL part.

Conversion Details

When you save a part data in XML format, the symbol and the chips information is written out in the XML file.

Translating Chips (Physical) View

The following information is translated from the Concept HDL part:

- Pin names
- Pin text
- Package properties
- Pin types
- Part aliases (equivalent packages)
Pin Names

The pin name on Concept HDL symbol pin is translated into XML as pin name. The Concept HDL assertion character is translated as per the Capture assertion mechanism where \ follows each character for low asserted pins.

Pin Text

The pin text is added as PIN_TEXT property to the ports in XML.

Note: The Pin_Text property may not necessarily exist in all Concept HDL parts. In the symbols generated by Part Developer and Concept HDL prior to the 14.0 release, the pin text on symbol pins was a plain text placed near the location of the pin. It is therefore possible, that the automated translation may not be able to locate this pin text by position as the pin text may not exist near the end of pin stub. In such cases, the pin text may not be found or incorrectly linked to another text in the search region. If the pin text is not found, the translation automatically uses the pin names instead of pin text while naming the XML pin. In the 14.0 release, Concept HDL and Part Developer have a property on the symbol pin called PIN_TEXT that links in the exact pin text so as to avoid ambiguity. It is recommended that the old symbols (created in releases prior to the 14.0 release) are saved in Part Developer and verified before converting them to XML. Part Developer links the pin text to pin by moving the text into a PIN_TEXT pin property. If this is not done then you may need to edit the translated XML part to remove the texts that were placed as texts next to pins but did not get detected as pin texts as they were not as PIN_TEXT property.

Package Properties

BODY_NAME is deleted in the XML export. All other properties are added either as properties or under a special XML tag if the DTD has a special provision for the value of that property. An example of a property that goes under a XML tag is the REFDES_PREFIX.

Pin Type

The pin types are converted as per the following table:

<table>
<thead>
<tr>
<th>Part Developer Pin Types</th>
<th>XML Direction</th>
<th>XML Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>Input</td>
<td>Input</td>
</tr>
<tr>
<td>OC</td>
<td>Output</td>
<td>OC</td>
</tr>
<tr>
<td>OE</td>
<td>Output</td>
<td>OE</td>
</tr>
</tbody>
</table>
### Part Alias

The aliases are added as a series of property PACKAGE_ALIAS(n), where n is the sequence number.

### Translating Symbol (Logical) View

The following are translated from the symbol view:

- **Pin Names**
- **Pin Location**
- **Graphics - Symbol Shape**
- **Graphics - Pin Shapes**
- **Pin Properties**
- **Symbol Properties**

#### Pin Names

See [Pin Names](#) in the translation of the chips (physical) view for details.

#### Pin Location

Special care is taken to ensure that the location of pins is an exact match to that in the Concept HDL part. This allows the design translators to be based on graphical translation. If
a pin is located within a bounding box then it is converted as a zero-length pin and the pin shape is converted to graphic lines.

Concept HDL part has two type of pins in the body section of the chips.prt file: Power pins and NC pins. These are pins that are not located on Concept HDL symbols.

For NC pins in body section, the translation places a property on the symbol called NC.

For Power pins in body section, the translation adds each of these pins on the bounding box of XML symbol starting from the top left corner in such a way that no two pins on the part overlap at hotspot. These pins are added as invisible pins.

### Graphics - Symbol Shapes

The following graphics entities are translated:

- Line
- Rectangle
- Ellipse
- Arc
- Polyline
- Filled shapes

### Graphics - Pin shapes

Concept HDL does not support pin shapes. The pin shapes in Concept HDL can consist of one or more graphics such as line, arc, circle and so on. While translating to XML, if a shape exists that matches to a Capture pin shape, then it is translated into equivalent Capture shape which is how the pins shape templates are represented in XML. For more details, see Capture User Guide.

The bubbled pins are translated as pin line shape in XML.

### Pin Properties

All pin properties are translated as is.
Symbol Properties

All pin properties are translated as is.

Limitation

The XML output from a Concept HDL part defaults the font information for all texts to the default in Capture.
Concept HDL Cell - Capture Part Translations

Overview

Note: This feature is only available in the Part Developer Expert version. This version is available with the PCB Librarian Expert licence.

Concept HDL and Capture are schematic editors. However the component library structure followed by Concept HDL and Capture are different. Therefore, there is a need to have the ability to convert between Concept HDL and Capture parts. The PSD 14.2 release of Cadence tools provides you the ability to convert between Concept HDL and Capture parts. You can convert between Concept HDL and Capture using either Part Developer or Project Manager.

To understand the conversion process, a reading of the Capture User Guide and the Concept HDL documentation is recommended.

Note: In the current release, only the physical information is translated between Capture and Concept HDL parts. Translation of simulation views is not supported in this release.

Note: This feature is available only on Windows NT.

Capture to Concept HDL Conversion

Overview

This translation path converts the Capture symbol into Concept HDL cells with the chips and symbol views. All the symbols (Normal and Convert) are translated.

The following types of Capture symbols are translated

- Normal Parts (Homogeneous/Heterogeneous)
Library Level Symbols (Power/Off-pageconnectors/Hierarchical/Title)

Note: The attached implementation is not translated by the tool.

Caution

If a Capture part has a pin number as 0, the pin gets converted to pin number zero in Concept HDL part. This is because Concept HDL does not support 0 as a pin number.

Caution

If a Capture Part has some of the pin numbers missing then it will create an incomplete chips file due to missing data in PIN_NUMBER property. You need to manually edit such a chips file to make it work with Concept HDL. For example, in the 74LS30 part covered later in the examples section, notice that pin numbers 7, 9 and 10 are missing. If you were to convert such a part, then the resulting chip file will not have the pin numbers 7, 9 and 10. To ensure that they appear in the chips file, you should add a user property called NC and add the pin numbers 7, 9 and 10 as its value. This causes the pins 7, 9 and 10 to appear as NC pins in the chips file.

In both the above mentioned cases, it is recommended that the Capture part is corrected before proceeding with translation.

Steps

1. Launch Part Developer.

2. Select File > Import > Capture Part.

   The Import from Capture to Concept HDL dialog box appears.

3. Browse and select the Capture library.

4. Select the Capture part from the Part drop-down list box.

   Note: If the selected part has aliases, they get displayed in the Aliases list box. By default, only the master components are shown in the Part drop-down list box. Therefore, in case you do not find a part listed in the drop-down list box, you need to find its master component and then convert the part.

5. Select whether you want to convert only the master component, or each alias as an individual primitive or each alias as an individual cell.
If you choose to convert only the master component, then one Concept HDL part is created. The chips.prt file will have only one primitive section with the master component name as its only entry.

If you choose to convert each alias as individual primitives, then one Concept HDL part is created with each alias appearing as a primitive entry in the chips.prt file. For example, if a part has four aliases, the resultant chips.prt file will have four primitives.

If you choose to convert each alias as an individual cell, then as many number of parts as there are aliases will be created. For example, if a part has four aliases, then four separate Concept HDL parts will be created.

6. Select the Concept HDL library in which you want to save the converted part and click OK.

Conversion Details

Conversion of the Chips (Physical) View

The following information is translated from the Capture part:

- Pin Names
- Pin Numbers
- Package Properties
- Pin Types
- Part Aliases

Pin Names

The pin names are translated into Concept HDL as pin names in the chips view and as pin text on the symbol pins in symbol view.

Due to restrictions in Concept HDL on certain characters, the following characters are replaced:

- > is replaced with _GT_
- < is replaced with _LT_
- ‘ is replaced with _

In case of duplicate pin names
If all the pins are visible and are of type power, then the pin is created as vectored pin in Concept HDL.

If all pins are not of type power, then all non-power pins which have duplicate pin names are converted to unique pin names in Concept HDL by appending \_<number> suffix to the pin name. The numbering starts from 1. For example, if the Capture part had 5 pins with the name A, then they will be converted to A_1, A_2, A_3, A_4, A_5 pins in Concept HDL.

If all the pins are invisible and are of type power, then all the pins are written in body section of the chips file in Concept HDL.

If a part has duplicate pins of type power and some pins are visible and some are not, then for visible pins, \_<number> is appended and the invisible pins are moved to body section in chips view in Concept HDL.

The low assertion character (* or _N) in the pin names is set on the basis of the settings in Concept HDL project file.

Pin Numbers

If the Capture part has no physical pin numbers - then no chips view is created in Concept HDL.

Note: Please see the Caution section above for additional information on pin numbers.

Package Properties

The properties translated are as follows:

- PCB footprint to JEDEC_TYPE
- Part Reference Prefix to PHYS_DES_PREFIX
- Name to Cell name and PART_NAME

Note: A Concept HDL part has capability to support additional data on pins such as pin loads. This information is not present in Capture and is therefore seeded with the values set in the project settings. The size property is not transferred as it has no relevance in Concept HDL domain.

Pin Type

The pin types are converted as per the table listed below:

<table>
<thead>
<tr>
<th>Capture Pin Types</th>
<th>Concept Pin Type</th>
</tr>
</thead>
</table>

January 2002 112 Product Version 14.2
Part Alias

As discussed in the steps, you have the choice to retain or drop the Capture part aliases. If aliases are to be retained, you have a choice to create them as packages in the same chips view or create them as different cells.

Conversion of Symbol View (Logical)

The following symbol information is translated from a Capture symbol:

- Pin Names
- Pin Location
- Graphics - Symbol Shape
- Graphics - Pin Shapes
- Pin Properties
- Symbol Properties

Pin Names

- The pin names are matched to the names in chips view.

Pin Location

- Special care is taken to ensure that the location of pins is an exact match to that in Capture part. This allows the design translators to be based on graphical translation. Please refer to the caution if you find a pin missing.
Graphics - Symbol Shapes

The following graphics entities are translated

- Line
- Rectangle
- Ellipse
- Arc
- Polyline
- Filled shapes

**Note:** The bit maps are not translated as they are not supported in Concept HDL. The filled shapes are shown as unfilled in Concept HDL as Concept HDL does not have support for filled shape.

**Note:** The fonts in a Capture symbol are not translated as Concept HDL does not support fonts.

Graphics - Pin shapes

- The Capture pin shapes are translated into Concept HDL to their equivalent graphics.

Pin Properties

The following properties are not transferred from Capture symbol ports:

- Name
- Number
- Order
- Long
- Clock
- Dot
- Pin_Number_Is_Visible
Symbol Properties

- If the property has no value in Capture, it is not transferred to the converted Concept HDL part.
- Each space in a property is converted to an ‘_’.
- The following property names and case are modified -
  - Implementation Path to IMPLEMENTATION_PATH
  - Implementation Type to IMPLEMENTATION_TYPE
  - Implementation to IMPLEMENTATION
  - PSpiceTemplate to PSPICETEMPLATE
  - NC to NC_PINS (this symbol property is transferred to chips view)

- The following properties are not added to Concept HDL symbol from Capture symbol
  - Name
  - Value
  - Part_Reference
  - Reference

Examples

The following Capture parts are used to demonstrate the conversion of Capture parts to Concept HDL parts.

- G6HK-DL-DPDT
- 74LS30
- 74LS241
G6HK-DL-DPDT

The graphical view of the part is displayed below:

```
    LS?
   9
  8
   7
   2
  3
   4
  5
   6
  1
 10
```

Notice that while the pin numbers are visible on the graphical symbols, the pin names are not visible. When this part is converted into Concept HDL, the symbol will show the pin names and not numbers. For example, the pin name for pin number 8 is COMMON2. When this part is converted, the Concept HDL symbol will show COMMON2 instead of 8.

The user properties are displayed below:
The part properties are displayed below:
After conversion, the symbol of G6HK-DL-DPDT part is displayed in Concept HDL.

**Note:** The NC pins in the symbol above are not the NC pins as it is understood in Concept HDL. These are actual pin names which have been translated from the Capture part.
The symbol properties in the converted part is displayed below:

![Symbol Properties](image-url)

**Note:** The symbol properties such as `Pin_Names_Visible` has no meaning in Concept HDL. However, these properties are retained while converting from Capture to Concept HDL to ensure that data is not lost in case the part is converted back to Capture.
**74LS30**

The graphical symbol of the Capture part is displayed below:

![74LS30 Symbol](image)

The user properties for the part is displayed below:

![User Properties](image)
The part properties for the part is displayed below:

![Edit Part Properties](image1)

The part has the following aliases:

![Part Aliases](image2)

During conversion, the *Each alias as an individual primitive* option was chosen. This ensures that all the aliases appear in the *chips.prt* file as a separate primitive entry.
Part Developer User Guide
Concept HDL Cell - Capture Part Translations

Capture To Concept HDL

Capture
- Library: D:\lib\hw\Gate.lib
- Part: 74LS30
- Options:
  - Master component alone
  - Each alias as individual primitive
  - Each alias as individual cell

Concept HDL
- Library: my_library

[OK]  [Cancel]  [Help]
The symbol view of the part after being converted into Concept HDL is displayed below:
A partial listing of the `chips.prt` file is displayed below. Notice that the user property NC gets converted to the `NC_PINS` property in the body section. Also notice that each alias is appearing as a separate primitive.

```plaintext
FILE_TYPE=LIBRARY_PARTS;
primitive '74LS30';
  pin
    'I0':
      PIN_NUMBER= (1);
      INPUT_LOAD= (-0.01,0.01);
    'I1':
      PIN_NUMBER= (2);
      INPUT_LOAD= (-0.01,0.01);
    'I2':
      PIN_NUMBER= (3);
      INPUT_LOAD= (-0.01,0.01);
    'I3':
      PIN_NUMBER= (4);
      INPUT_LOAD= (-0.01,0.01);
    'I4':
      PIN_NUMBER= (5);
      INPUT_LOAD= (-0.01,0.01);
    'I5':
      PIN_NUMBER= (6);
      INPUT_LOAD= (-0.01,0.01);
    'I6':
      PIN_NUMBER= (11);
      INPUT_LOAD= (-0.01,0.01);
    'I7':
      PIN_NUMBER= (12);
      INPUT_LOAD= (-0.01,0.01);
    'Q':
      PIN_NUMBER= (8);
      OUTPUT_LOAD= (1.0,-1.0);
  end_pin:
  body
    NC_PINS= (9,10,13);
    PHYS_DES_PREFIX= 'U';
    CLASS= 'IC';
    PART_NAME= '74LS30';
    POWER_PINS= 'VCC:14';
    POWER_PINS= 'GND:7';
  end_body;
end_primitive;
primitive '74LS30 FP';
  pin

LS241

The Capture symbol of the LS241 part is displayed below:
The part properties are displayed below:

This part when converted into Concept HDL will have two symbols, one for each group. Each group will have four sections.

The symbol views of the part converted into Concept HDL is displayed below:
Notice that in Sym_2, the $\overline{OE}$ pin is written out at $O\backslash E\backslash$. 
The `chips.prt` file of the LS241 is displayed below:

```
FILE_TYPE=LIBRARY_PARTS;
primitive='74LS241';
  pin
    'A':|
    PIN_NUMBER='(17,15,13,11,0,0,0,0)';
    INPUT_LOAD='(-0.01,0.01)';
    'Y0':
    PIN_NUMBER='(3,5,7,9,0,0,0,0)';
    INPUT_LOAD='(-0.01,0.01)';
    OUTPUT_LOAD='(1.0,-1.0)';
    OUTPUT_TYPE='(TS,TS)';
    'OE':
    PIN_NUMBER='(19,19,19,19,0,0,0,0)';
    INPUT_LOAD='(-0.01,0.01)';
    '-OE':
    PIN_NUMBER='(0,0,0,1,1,1,1)';
    INPUT_LOAD='(-0.01,0.01)';
    'B':
    PIN_NUMBER='(0,0,0,3,6,4,2)';
    INPUT_LOAD='(-0.01,0.01)';
    'Y1':
    PIN_NUMBER='(0,0,0,12,14,16,18)';
    INPUT_LOAD='(-0.01,0.01)';
    OUTPUT_LOAD='(1.0,-1.0)';
    OUTPUT_TYPE='(TS,TS)';
  end_pin;
body
  PHYSICAL_PREFIX='U';
  CLASS='IC';
  PART_NAME='74LS241';
  POWER_PINS='(VCC:20)';
  POWER_PINS='(GND:10)';
end_body;
end_primitive;
END.
```

Note that LS241 has 8 parts per package. This gets converted to 8 slots in the `chips.prt` file.

**Concept HDL to Capture Conversion**

**Overview**

The Concept HDL to Capture conversion is essentially the conversion of the symbol and chips view of a Concept HDL cell to a Capture Part.

The following types of parts are converted:
- Normal flat-parts
- Sized parts. Such parts are converted to homogeneous parts in Capture.
- Split parts. Such parts are converted to heterogeneous parts in Capture.
- Asymmetrical parts. Such parts are converted to heterogeneous parts in Capture.

**Note:** The translation does not support the creation of library level symbols such as title blocks and power symbols, into Capture database.

**Note:** Capture database supports a maximum of two symbols for each function in a part. Therefore, the user can choose a maximum of two symbols per group. Also in order to have the full functionality of the part to be translated, the user must choose at-least one symbol for each group. There is no equivalent of FIXED_SIZE symbols in Capture. Therefore, these symbols are not translatable.

**Steps**

1. Launch Part Developer.
2. Select *File > Export > Capture Part*.
   
   The *Export from Concept HDL to Capture* dialog box appears.
3. Select the Concept HDL library and part that you want to convert to Capture and click *OK*.
The *Export from Concept HDL to Capture - Select symbols* dialog box appears.

4. Select the package that you want to convert from Concept HDL to Capture.

5. After you select the package, you need to select the symbol(s) from the list of symbols. You can select a maximum of two symbols for conversion.

**Note:** Only those symbols that can be packaged into the selected package are displayed. Also, since Capture does not support the notion of symbols with fixed size, none of the Concept HDL symbol that have the HAS_FIXED_SIZE property will appear in the list of symbols.

6. If you want to use the symbol port names to represent the pins in Capture, select *Use Pin Name to write Capture Port Name*. Otherwise, the value of the PIN_TEXT property is used to represent the pin names in Capture. For example, a Concept HDL symbol has PIN_TEXT property value as ABC_CLOCK for the symbol pin A. If you select the *Use Pin Name to write Capture Port Name* option, then the value of Pin Name property in Capture for symbol pin A will be A. If the option is not selected, the value of the Pin Name property in Capture will be ABC_CLOCK.

**Caution**

*Parts created in Concept HDL or versions of Part Developer prior to 14.0 do not have the PIN_TEXT property. When such parts are converted with the Use Pin Name option switched off, the tool checks for any text*
towards the right of the pins. If it finds such text, it uses that text as the pin name. If it fails to find any text, it uses the symbol port name as the Pin Name. Therefore, it is important that you check the symbols after they are converted from Concept HDL to Capture.

7. Select the Capture library where you want to store the converted part and click OK.

Note: In case the tool fails to find the Capture library in the specified location, it creates the new library automatically.

Conversion Details

Translation of the Chips (Physical) View

The following information is translated from the chips view of a Concept HDL part:

- Pin Names
- Package Properties
- Pin Types
- Part Aliases (equivalent packages)

Pin Names

The pin text on Concept HDL symbol pin is translated into Capture as pin name. Alternatively, you can also select Concept HDL part’s pin name instead of pin text.

Note: In the Concept HDL part, the pin text may not necessarily exist. In the symbols generated by Part Developer and Concept HDL prior to 14.0, the pin text on symbol pins was a plain text placed near the location of the pin. It is therefore possible, that the automated translation may not be able to locate this pin text by position as pin text may not exist near the end of pin stub. In such cases, the pin text may not be found or incorrectly linked to another text in the search region. If the pin text is not found, the tool automatically uses the pin names instead of pin text while naming the Capture pin. In 14.0, Concept HDL/Part Developer has a property on the symbol pin called PIN_TEXT that links in the exact pin text so as to avoid the ambiguity. It is recommended that the old symbols (created in releases prior to 14.0) are saved in Part Developer and verified before being converted to Capture. Part Developer associates the pin text to pin by moving the text into a PIN_TEXT pin property. If this is not done then you may need to edit the translated Capture part to remove the texts that were placed as texts next to pins but did not get detected as pin texts as they were not as PIN_TEXT property.
If the pin name is used instead of pin text then the following characters are replaced:

- \_GT\_ to >
- \_LT\_ to <

The sizable ports in the Concept HDL sizable part are converted to scalar ports of the converted homogeneous part.

The vector bits are converted as scalar ports. For example, A[10] is converted to Capture pin A10. However, a vector bus is converted as is.

### Package Properties

Concept HDL parts have the ability to support additional data on a pin such as pin loads. Such information is added to the symbol pins on Capture.

The package properties are translated as follows:

- JEDEC_TYPE to PCB footprint
- BODY_NAME to Value as Capture Symbol property
- NC_PINS to NC as Capture Symbol Property
- PHYS_DES_PREFIX to Part Reference Prefix

### Pin Type

The pin types are converted as per the following table:

<table>
<thead>
<tr>
<th>Concept Pin Type</th>
<th>Capture Pin Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>INPUT</td>
</tr>
<tr>
<td>Output</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>OC</td>
<td>Open Collector</td>
</tr>
<tr>
<td>OE</td>
<td>Open Emitter</td>
</tr>
<tr>
<td>TS</td>
<td>3-State</td>
</tr>
<tr>
<td>BIDIR</td>
<td>Bidirectional</td>
</tr>
<tr>
<td>POWER</td>
<td>POWER</td>
</tr>
<tr>
<td>UNSPEC</td>
<td>PASSIVE</td>
</tr>
<tr>
<td>GROUND</td>
<td>POWER</td>
</tr>
<tr>
<td>ANALOG</td>
<td>PASSIVE</td>
</tr>
<tr>
<td>OC_BIDIR</td>
<td>Bidirectional</td>
</tr>
</tbody>
</table>
Part Alias

- The equivalent packages in a Concept HDL part (the list of packages in single primitive line in chips view) are made as aliases in the Capture part.
- The last character, _, is changed to / in the alias names in accordance with Capture standards for naming aliases.

Translating Symbol (Logical) View

The following information is translated from the Concept HDL symbol:

- Pin Names
- Pin Location
- Graphics - Symbol Shape
- Graphics - Pin Shapes
- Pin Properties
- Symbol Properties

Pin Names

See section on pin names in translating from the chips view.

Pin Locations

Special care is taken to ensure that the location of pins is an exact match to that in Concept HDL part. This allows the design translators to be based on graphical translation. If a pin is located within a bounding box (which is not in accordance with Capture standards) then it is converted as a zero-length pin and the pin shape is converted to graphic lines.

A Concept HDL part can have two type of pins in the body section: Power pins and NC pins. These are pins that are not located on the Concept HDL symbols.

For NC pins in body section, the translation places a property on the symbol NC.
For Power pins in body section, the translation adds each of these pins on the bounding box of Capture starting from the top left corner in such a way that no two pins on the part overlap at hotspot. These pins are added as invisible pins.

**Note:** If a translated part has pins within the bounding box and the symbol is edited in Capture then symbol editor automatically moves the pins out to the bounding box.

**Graphics - Symbol Shapes**

The following graphics entities are translated:

- Line
- Rectangle
- Ellipse
- Arc
- Polyline
- Filled shapes

**Graphics - Pin shapes**

Concept HDL does not support pin shapes. The pin shapes in Concept HDL can consist of one or more graphics such as line, arc, circle and so on. In translation to Capture, if a shape exists that matches to a Capture pin shape, then it is translated into equivalent Capture shape.

The bubbled pins are translated as pin line shape in Capture. All pins that are on or inside the calculated bounding box are converted to zero length pins.

**Pin Properties**

All pin properties are translated as is.

**Symbol Properties**

The following property names and case are modified:

- IMPLEMENTATION_PATH to Implementation Path
- IMPLEMENTATION_TYPE to Implementation Type
IMPLEMENTATION to Implementation

PSPICETEMPLATE to PSpiceTemplate

NC_PINS to NC. This symbol property is transferred from chips view.

Examples

The conversion of Concept HDL parts to Capture is displayed using three parts:

- LS00
- LS241

**LS00**

LS00 is a symmetrical part with four sections. It has two symbols, sym_1 and sym_2.

The symbol stored in sym_1 is displayed below:

![Symbol for sym_1](image)

The symbol stored in sym_2 is displayed below:

![Symbol for sym_2](image)
The partial `chips.prt` file for LS00 is displayed below:

```plaintext
FILE_TYPE=LIBRARY_PARTS;
TIME=' COMPILATION ON THU JAN 10 14:52:02 1991 ';
primitive '74LS00','74LS00_DIP';
  pin
   'B'<>:
     INPUT_LOAD='(-0.4,0.02)';
     PIN_NUMBER='(13,10,5,2)';
     PIN_GROUP='1';
   'A'<>:
     INPUT_LOAD='(-0.4,0.02)';
     PIN_NUMBER='(12,9,4,1)';
     PIN_GROUP='1';
   '-Y'<>:
     OUTPUT_LOAD='(8.0,-0.4)';
     PIN_NUMBER='(11,8,6,3)';
end_pin;
body
  POWER_PINS='(VCC:14;GND:7)';
  FAMILY='LSTTL';
  PART_NAME='74LS00';
  BODY_NAME='LS00';
  DEFAULT_SIGNAL_MODEL='SN74LS00N  TI';
  JEDEC_TYPE='DIP14_3';
  CLASS='IC';
  TECH='74LS';
end_body;
end_primitive;
primitive '74LS00_SOIC';
  pin
   'B'<>:
     INPUT_LOAD='(-0.4,0.02)';
```

January 2002

Product Version 14.2
The second symbol after being converted to Capture is displayed below:

This is also called the normal view of the part. There are four symbols because there were four sections in the `chips.prt` file. Each section is converted as a part in the package. Note that all the symbols have two circles near pin A. These denote the pins that were on the body section of the `chips.prt` file of the Concept HDL part. These pins have zero length and are invisible.
LS241

LS241 is an asymmetrical part with two groups. Each group has four section each. The symbols for both slot groups is displayed below:

![Diagram of LS241](image1)

![Diagram of LS241](image2)
The partial listing of the chips.prt file is displayed below:

```plaintext
FILE_TYPE=LIBRARY_PARTS;
TIME='Created/Modified on Wed Sep 19 13:26:34 2001';
primitive '74LS241','74LS241_DIP';
  pin
    'Y1'<O>:
      PIN_NUMBER='(12,14,16,18,0,0,0,0,0)';
      INPUT_LOAD='(-0.01,0.01)';
      OUTPUT_LOAD='(1.0,-1.0)';
      OUTPUT_TYPE='(TS,TS)';
    'B'<O>:
      PIN_NUMBER='(8,6,4,2,0,0,0,0,0)';
      INPUT_LOAD='(-0.01,0.01)';
    '-OE1':
      PIN_NUMBER='(1,1,1,1,0,0,0,0,0)';
      INPUT_LOAD='(-0.01,0.01)';
    'YO'<O>:
      PIN_NUMBER='(0,0,0,0,3,5,7,9)';
      INPUT_LOAD='(-0.01,0.01)';
      OUTPUT_LOAD='(1.0,-1.0)';
      OUTPUT_TYPE='(TS,TS)';
    'OE0':
      PIN_NUMBER='(0,0,0,0,19,19,19,19)';
      INPUT_LOAD='(-0.01,0.01)';
    'A'<O>:
      PIN_NUMBER='(0,0,0,0,17,15,13,11)';
      INPUT_LOAD='(-0.01,0.01)';
  end_pin;
body
  POWER_PINS='(VCC:20;GND:10)';
  CLASS='IC';
  PART_NAME='74LS241';
  JEDEC_TYPE='DIP20_3';
  FAMILY='LSTTL';
  BODY_NAME='LS241';
  DEFAULT_SIGNAL_MODEL='SN74LS241N TI';
  TECH='74LS';
end_body;
end_primitive;
```
The part after conversion to Capture is displayed below:

Notice that each of the eight sections of LS241 in Concept HDL has converted into a part in Capture.

The part properties of the LS241 in Capture is displayed below:
Notice that the JEDEC_TYPE property gets converted to the PCB Footprint property in Capture.
Verifying Parts

Overview

After you complete the creation of a part, you can run various checks on it to ensure that they are correct. To know more about the checks, see Appendix E, “Checks.”

To run checks:

1. Select Tools > Verify.
2. Select the required check.
3. Click OK.

Note: In case you have a part which has names with “(,” “)” or “&” symbol, you should run the hlibftb library checking utility from the command line with the -advopt option.

You can run the following checks:

- View Verification
- Instantiation and Packaging
- Advanced View Checks
- VHDL Compilation
- Verilog Compilation
- Verify with Templates

View Verification

You can run the following checks:
Part Developer User Guide
Verifying Parts

- Symbol origin is centered.
  Checks whether the origin always lies within the symbol and the symbol is at a distance less than the maximum allowed offset from the origin.

- Tristated pins have input and output loads defined.
  Checks the presence of pin properties OUTPUT_LOAD and INPUT_LOAD for every tri-state pin. This is denoted by the property OUTPUT_TYPE =TS,TS.

- Mandatory properties present in package file.
  Checks whether the properties named BODY_NAME, PART_NAME, CLASS, and JEDEC_TYPE are present in the chips.prt file.

- Consistent symbol name in symbol and package file.
  Checks whether the symbol text is the same as BODY_NAME in the chips.prt file.

- Consistent symbol and package in pin list.
  Checks whether the pins are the same across symbol and package views.

Instantiation and Packaging

Instantiation and Packaging checks include the following:

- Use Project ptf files for verification
  If you select this rule, part table files are used in instantiation and packaging.
  If no part table files are specified in the project file, the cell-level ptf is used by default.

- Use allegro board (netrev)
  If you select this option, the part or library is verified for the complete Front to Back flow.

- Generate Pass/Fail report
  This option is enabled only if you select more than one part or when you select a library that has more than one part.
  **Note:** If you select the Generate Pass/Fail option, each part is verified separately. This is a time consuming process.

Advanced View Checks

Select this option to launch CheckPlus. You can run your own custom-defined checks using CheckPlus.
VHDL Compilation

Use this option to compile the generated VHDL wrapper. You can use either NCVHDL or CV to compile the wrapper. You can specify the tool to compile the VHDL wrapper in the Enter command in the VHDL compilation dialog box. This dialog box is displayed when you click the Options button in the Verifications dialog box.

Verilog Compilation

Use this option to compile the generated Verilog wrapper. You can use NCVERILOG to compile the wrapper. You can specify the tool to compile the Verilog wrapper in the Enter command for Verilog compilation dialog box. This dialog box is displayed when you click the Options button in the Verifications dialog box.

Verify with Templates

Select this option to verify a selected part against a template. The result of the verification is displayed in a report. The verification is done as per the following rules:

Property Checks

The property check is done on all packages for the following:

- All properties listed in the template for a package must exist in each package of the part
- The value of the property in each package must match the value in the template unless the value in template is "?" or blank.

Pin Load Checks

This is done on all pins in all packages as per the following rules:

- If PINUSE="UNSPEC" exists for a pin, all checks are bypassed on that pin
- If a pin type is not determined, it is an error
- If a pin type is determined, its load value is checked against the load value of that pin type in the template. An error is generated if the load values don’t match.
- Error is shown if any of the loads for a pin type is missing
Symbol Checks

All symbols are checked for a given part as per the following rules:

- All symbols must have at least one connection with a line stub or a bubble else an error stating that check cannot proceed is shown.
- All lines are assumed to be vertical or horizontal. Arcs are not supported in this release.
- Each Bubble is interpreted to have two virtual stub lines - horizontal and vertical.
- No two connections can have the same X, Y coordinates else error is shown to the user.
- Location of connections is derived based on the direction of the stubs attached to the connection. Therefore, only connections that have a stub or a bubble on them are checked.
- Stub length is calculated based on the integer average of all stub lengths.
- The outline of the body is derived by searching for the perpendicular line from the end of stub. As the stub size varies, the search is made within the range of the stub size variance. After getting a single outline, the rest are traced as the ones connected the outline end points. The procedure is executed recursively for each detected outline.
- Minimum pin spacing are calculated for all sides (Top, Left, Right, Bottom).
- For pin texts, the property PIN_TEXT is used. If it is not found, pin texts are searched for within 1/3 of the average stub length from the end of the stub for each connection. Also the location (x,y) of the pin note must not be mis-aligned by more than 1/2 pin spacing.
- Grid is derived by taking the highest common factor of all differences of values of X and Y on respective coordinates. Only the connection (Logic) grid is derived. The symbol grid is not derived even though the template mentions it as Symbol Grid.
- All properties listed in the template for symbols must exist on each symbol in the part.
- The value of the property in each symbol must match the value in the template unless the value in template is "?" or blank.
- The alignment of the property must match the one specified in template.
- The visibility of the property must match the one specified in template.

Pin Checks

Each pin is checked as per the following rules:
Each pin based on the type as defined in the template must be at the location area in symbol as defined in the template for that type.

- The text size of the pin must match the size specified in the template.
- The *Use Pin Names For Text* is checked only if the template sets it to true.
- The text style for pin text is checked with value in template. If the style is vertical for top and bottom pins and horizontal for pins on left and right then the style is considered to be Automatic. The angles of 90 and 270 are considered equivalent and vertical and 0 and 180 are considered equivalent and horizontal.
- All pins with spacing less than the spacing specified in the template are marked as errors.

### Grid Checks

Grid checks are done with the following rules:

- Conversions are done for calculating and matching the grid values for Inches, Metric and Fractional (Fractional is currently not supported in Part Developer). This is then matched with the template value.

### Outline Checks

The outline checks are done with the following rules:

- All detected Outlines are checked to match the thickness specified in template.

### Minimum Size Checks

- The minimum pin spacing values on the left and right is read for each symbol and verified against the minimum pin spacing left and right value stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.
- The minimum pin spacing values on the top and bottom is read for each symbol and verified against the minimum pin spacing top and bottom value stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.
- The minimum symbol height value is read for each symbol and verified against the minimum symbol height stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.
The minimum symbol height width is read for each symbol and verified against the minimum symbol width stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.

The output of the verification is displayed in a dialog box. The output is divided into two sections, Overview and Details.

In the Overview section, the overview of the differences are displayed. In the details section, the differences are detailed.
Example - Symmetrical Part Creation

Overview

A part in which the logical pin list is the same across packages and slots is known as a symmetrical part. The generic component LS00 (a quadruple 2-input positive-nand gates symmetrical part) and its variations such as the SN5400, SN54LS00, SN54S00, SN7400, SN74LS00, and SN74S00 components will be used as an example while detailing the steps to create a symmetrical part. This will be created in a build library named my_library. This part has four independent 2-input NAND gates and comes in DIP, SOIC, Ceramic Chip Carrier (CCC) and Ceramic Flat Packages (CFP). The pinouts for the specific packages are displayed below:

Note: In the diagram displayed above, the following representations are used for the packages.

<table>
<thead>
<tr>
<th>J Package</th>
<th>DIP Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>W Package</td>
<td>Ceramic Flat Package (CFP)</td>
</tr>
<tr>
<td>N Package</td>
<td>SOIC Package</td>
</tr>
</tbody>
</table>
Setting Up the Part Developer Defaults (Example of a Symmetrical Part)

1. Specify the default options for the part. Because the pin name is represented in LS00 as &lt;Numeral&gt;&lt;Alphanumeric&gt; and the part has four slots, select the Slot-Base Name option in the Pin Name Interpretation tab in the Part Developer Setup Options dialog box.

The next step is to enter the part properties such as the part name, and the logical pin information.

Entering Part Properties and Logical Pins Information (Example of a Symmetrical Part)


   The New Part Properties dialog box appears. You need to specify the properties for the new part in this dialog box.

2. Select the library name as my_lib.

3. Specify the part name as LS00.
4. Specify the physical part name as **SN5400**.

5. Specify the part type. For LS00, select **IC**.

   The completed dialog box should look like this:

   ![New Part Properties dialog box](image)

   - **Library Name**: my_library
   - **Part Name**: LS00
   - **Physical Part Name**: SN5400
   - **Part Type**: IC

6. Click **OK**.
The *Logical Pins* dialog box appears.

7. Select the pin type as *INPUT*.

   **Note:** This is selected by default.

8. Enter the logical pin names for input pins as 1A–4A, 1B–4B.
9. Click *Add* to add the specified pin names to the logical pin list.

10. Repeat the step 8 and 9 for different types of pins for the part. For LS00, to enter output pins, select pin type as *OUTPUT*, enter $1Y-4Y$ as the pin names and click *Add*. Repeat for GND and VCC pins.

The loaded Logical Pins dialog box for LS00 should appear as displayed below:
11. Click OK to accept the entries of the Logical Pins dialog box.

The next step is to create the packages for the part.

Creating Packages (Example of a Symmetrical Part)

When you create a package, the chips view is created. The purpose of the chips view is to define the available package styles (SOIC, DIP etc.), logical to physical pin mapping, power requirements and the physical package properties, such as the Allegro package symbol name.

For the LS00 part:


   The New Package dialog box appears. The General tab is selected by default.

   **Note:** Alternately, the New Package dialog box can be displayed by the RMB menu option on a “Packages” tree item.

2. Select the Specify Package Type checkbox.

3. Enter the package type as DIP.

4. Enter the equivalent packages, as SOIC.

5. To enter the JEDEC_TYPE value, click Specify Footprint.

   The Physical Properties dialog box displays.

6. Enter ? as the JEDEC_TYPE value.

7. Click OK.
For SN5400, the General property sheet will be as follows:

![New Package Window](image)

Cell Name: Iso00
Physical Part Name: SN5400
Specify Package Type: DIP

Reference Designator Prefix:

Footprint Information:
- JEDEC_TYPE: ?
- ALT_SYMBOLS:

Additional Properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>


8. Click the *Physical Pin Mappings* tab. This tab enables you to add physical pins and map them to the logical pins.
Example - Symmetrical Part Creation

Package Properties

Logical Pins:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>2A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>3A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>4A</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>2B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>3B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>4B</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>1Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>2Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>3Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>4Y</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>POWER</td>
<td></td>
</tr>
<tr>
<td>VCC</td>
<td>POWER</td>
<td></td>
</tr>
</tbody>
</table>

Mapped/Total: 0/14

Physical Pins:

1 unmapped logical and 1 physical pins selected

Specify Physical Pins

- Extract from Footprint
- Add Manually

Physical Pin Selection: [ ] [Select Pins]
9. Choose the *Pin Numbers*... button to add the physical pins.

   The *Add Physical Numbers* dialog box appears.

![](image)

10. Specify the physical pins. For SN5400, add 1–14 as Numeric physical pin numbers.

11. Click *OK* to accept your entries.

   This lists all the entered physical pins under the Physical Pins list in the Physical Pin Mapping property sheet.
12. Choose the logical and physical pin you want to map, and click Map. The mapping will depend on the pinout diagram as detailed in the datasheets. For example, for the DIP package of SN5400 component, the mappings are displayed below:
13. Click OK to complete the creation of the package.
Similarly, create other packages for the generic LS00 part.

Creating Symbols (Example of a Symmetrical Part)

To enable you to create symbols effectively, Part Developer provides a Symbol Creation Wizard. The Symbol Creation Wizard guides you through the process of creating symbols.

For the LS00 component, because there are NC pins on the body of the SN54LS00 part, the wizard asks you to specify whether you want the power and NC pins on the symbol. If you want to add power and NC pins to the symbol, go to Creating Symbols with Power / NC Pins on the Symbol on page 159. If you want to create symbols without the power or the NC pins, go to Creating Symbols without Power/NC Pins on Body on page 164.

Creating Symbols with Power / NC Pins on the Symbol

The Symbol Creation Wizard appears.


3. Click Next.
This displays the Package Selection page of the Wizard. This page is displayed because the SN5400 DIP package does not have NC pins whereas the SN54LS00 CCC package has NC pins.

4. To base the symbol on the SN54LS00 CCC package, choose SN54LS00_CCC.

5. Click Next.
Part Developer User Guide
Example - Symmetrical Part Creation

This displays the Symbol Kinds page.

6. To create a symbol with logical pins and a symbol with logical and NC pins, check the Logical and Logical + NC check boxes.

**Note:** Notice that there are options in the Symbol Kinds page that are separated by either by commas or a + sign. The difference between the two types is that if you select an option with comma, then two different symbols are created. For example, if you select the Logical, NC check box, then two symbols, one with logical pins and another with NC pins gets created. If you select an option with the + sign, then only one symbol gets created. For example, if you select the Logical + Power option, then only one symbol with both logical pins and power pins gets created.

7. Click Next.
This displays the Package Types page of the wizard.

![Symbol Creation Wizard - Package Types](image)

**Note:** The Package Types page of the Wizard enables you to specify the package type for the types of symbols you choose to create in the Symbol Kinds page. Depending on the symbols that you have chosen to create, you may face a situation where the existing packages may not be in synchronization with a symbol. For example, for the logical+NC symbol to be in synchronization with a package, the `chips.prt` file must have a primitive where the NC pins are in the pin section. However, the existing packages have the NC pins only in the body section of the `chips.prt` file. To resolve such situations, you need to create new packages. In this case, a new package needs to be created which will have the NC pins in its pin section. The new packages need to be sourced from the existing packages. You may choose to follow certain conventions while naming your packages, such as `partname_packagetype_L_NC` for logical+NC pins on the symbol.

8. To create the corresponding packages, for the Logical + NC symbol, specify the package type as `SN54LS00_CCC_L_NC` and the *Source Package* as CCC.
9. Choose Next.

   This displays the Summary page. Check the summary to ensure its correctness.

10. Choose Finish to create the symbols.

Creating Symbols without Power/NC Pins on Body

The Symbol Creation Wizard appears.

2. Choose No in the Power/NC Pin Selection page of the wizard.
This displays the Specify Part Name and Pack Type page.

![Symbol Creation Wizard - Specify Part Name and Pack Type](image)

**Note:** In this page you decide whether to base the symbol independent of a technology and a package or to make the symbol specific to a physical part and package. The decision whether or not to associate a symbol with a physical part or a package depends on whether the symbol is a representation of only one physical part or package. If the symbol is specific to a part or package, then you should associate the symbol with specific parts or packages. Also, if you intend to create more packages or physical parts that have different logical pin lists, then you should associate the symbols with their respective physical parts/packages.

3. Choose *Next*.

   This displays the *Summary* page. Check the summary to ensure its correctness.

4. Choose *Finish* to create the symbols.
Example - Asymmetrical Part Creation

Overview

Asymmetrical parts are parts in which the logical pin lists across packages or slots are different. An LS241 (octal buffers and line drivers with three state inputs) present in SN74LS241 is used as an example to detail the steps for creating an asymmetrical part. This part comes in DIP, SOIC, and CFP packages.
Setting Up the Part Developer Defaults (Example of an Asymmetrical Part)

The first step in creating an asymmetrical part is to set up the defaults for the part.

As displayed, the LS241 component is a 8-slot part with two slot groups, one with low asserted enable signal 1G* and one with high asserted enable signal 2G. In the datasheet the pins are present in one of the two formats:

■  <Numeral><Alphanumeric String><Numeral> such as 1A1 and 1A2 and in
    <Numeral><Alphanumeric String> such as 1G* and 2G where leading numeral denotes the group.
■  <Alphanumeric String><Numeral> such as A1, Y1, G and G*

You can enter the pin information in either the Group-BaseName-Slot notation which is the first one or in BaseName-Slot notation which is the second one. The following example is given using the Group-BaseName-Slot notation. In case you want to use the BaseName-Slot notation, see Creating Asymmetrical Parts using the Expand to Slots Option on page 177 for steps.

As per the datasheet, the leading numeral represent the group, the alphanumeric string represents the base name, and the trailing numeral represents the slot number. Therefore, the pins need to be interpreted as Group-Base Name-Slot and Group-Base Name respectively.

The steps are:

1. Choose Tools > Options.
   The Part Developer Setup Options dialog box appears.
2. Choose the Pin Name Interpretation tab.
4. Click OK.

The next step is to enter the part properties, such as the part name, and the logical pin information.
Entering Part Properties and Logical Pin Information
(Example of an Asymmetrical Part)

Entering part properties involves specifying the library name in which you want the part to be
stored, the name that you want to give the part, such as LS241, the physical part name such
as SN74LS241, and the type of part, such as IC.

For example, to enter the part properties for LS241:

   The New Part Properties dialog box appears.
2. Choose my_ library from the Library name drop-down list.
3. Enter the part name as LS241.
4. Enter the physical part name as SN74LS241.
5. Choose the part type as IC.
   Note: IC is the default option in the Part Type drop-down list.
6. Click OK.

This brings up the Logical Pins dialog box. You enter the logical pin information in this dialog
box.

For the part LS241:

1. Click View to ensure that the pin name interpretation is Group-BaseName-Slot for
   <Numeral><Alphanumeric String><Numeral> and Group-Base Name for
   <Numeral><Alphanumeric String>.
   Next, enter the input pins 1A1-1A4, 2A1-2A4, 1G and 1G*
2. Select the Pin Type as Input.
3. Enter the Pin Name as 1A1-1A4, 2A1-2A4.
4. Click Add.
5. Enter Pin Name as 1G.
6. Click Add.
7. Enter the Pin Name as 2G.
8. Select the Active Low check box.
9. Click *Add*.

10. Similarly, add the 1Y1-1Y4, 2Y1-2Y4 as the output pins and VCC and GND as the power pins.

The *Logical Pins* dialog box for LS241 part is displayed below:

![Logical Pins dialog box](image)

11. Click *OK* to accept the entries in the *Logical Pins* dialog box.

The next step is to create packages for the part.
Creating Packages (Example of an Asymmetrical Part)

Creating packages involves defining package types, equivalent packages, reference designator prefix and additional properties, if any, and specifying the logical to physical pin mappings. For more information, see

To create packages for the LS241 part:


2. Specify the Physical Part Name as SN74LS241.

   **Note:** SN74LS241 is the default entry.

3. Enter the package type as SOIC.

4. Click Specify Footprint.

   The Physical Properties dialog box appears.

5. Click Browse.

   The Footprints dialog box appears.

6. Select dip_20 as the footprint.

7. Click OK.

8. Click OK.

9. Select the Physical Pin Mapping tab.

10. Select the Extract from Footprint option.

11. Click Pin Numbers.

    This adds the physical pins to the Physical Pins list.

12. Select the logical pin and the physical pin that you want mapped and click Map. For example, choose 2 in the Physical Pins list and 1A1 in the Logical Pins list and click Map. This maps the Physical Pin 2 to the Logical Pin 1A1.

13. Repeat step 8 for all pins.
The filled-in Physical Pin Mapping sheet is as follows:
14. Click OK to accept the entries.

After the packages are created, the next step is to create the symbols.

Creating Symbols (Example of an Asymmetrical Part)

You launch the Symbol Creation Wizard (Choose File> New> Symbol) to ensure the error-
free creation of symbols. The Symbol Wizard is dynamic and, depending upon the pin
information, displays the appropriate options.

For example, for the LS241 package, the Symbol Wizard determines that there are 2 slot
groups. Therefore, it starts the process of creating symbols for both slot groups.

To create symbols for the LS241 part:

The Symbol Creation Wizard appears.

2. Click *Select All* to create symbols for both the slot groups.
3. Click *Next*. 
The Symbol Creation Wizard - Symbol Kinds page appears.

4. Select Logical (Symbol with only logical pins) to create a symbol with only logical pins.

5. Click Next.
The Symbol Creation Wizard - Specify Part Name and Pack Type page appears.

**Note:** In this page, you decide whether to base the symbol independent of a technology and a package or to make the symbol specific to a physical part and package. The decision whether or not to associate a symbol with a physical part or a package depends on whether the symbol is a representation of only one physical part or package. If the symbol is specific to a part or package, then you should associate the symbol with the specific part or package. Also, if you intend to create more packages or physical parts that have different logical pin lists, then you should associate the symbols with their respective physical parts/packages.

6. Check the *Specify PART_NAME property* check box to associate the symbol with the SN74LS241 part.

7. Check the *Specify PACK_TYPE property* to associate the symbol with the SOIC package type.
8. Click Next.

   The Symbol Creation Wizard - Summary page appears.

   ![Symbol Creation Wizard - Summary](image)

   Following are the symbols to be created:
   
   **sym_1**
   - PIns: A, G, Y
   - Associated Package: SN74LS241_SOIC

   **sym_2**
   - PIns: A, G*, Y
   - Associated Package: SN74LS241_SOIC

9. Click Finish after verifying the results in the summary.

Creating Asymmetrical Parts using the Expand to Slots Option

In parts such as LS241, if you choose to use the BaseName-Slot notation for entering the logical pins then there has to be a mechanism to determine to which slot groups do the scalar pins G and G* belong. This ability is provided by the Expand to Slots function. To do so:

1. Setup Part Developer (Tools > Options > Pin Name Interpretation) to interpret <Alphanumeric String><Numeral> logical pins as BaseName-Slot.

2. Enter the logical pins, as A1-A8, B1-B8, Y1-Y8, G, and G*.
**Note:** Note the difference in the logical pin naming. If instead of BaseName-Slot interpretation, the Group-BaseName-Slot would have been chosen, then the pins would have been entered as 1A1-1A4, 2A1-2A4, 1B1-1B4, 2B1-2B4, 1Y1-1Y4, 2Y1-2Y4, 1G and 2G*.

3. Create a new package.
4. Enter/extract the required physical pins.
5. Associate the logical pins to the physical pins except G and G*.
6. Right-click on G.
7. Choose *Expand to slots*.
   
   This will expand the scalar pin G across all the slots in the part. For LS241, the scalar pin G will get expanded to all the eight slots, G<1>..G<8>.

8. Similarly expand the other scalar pin G*.
9. Since G is common across the first four slots, map the physical pin 26 to G<1>..G<4>. Leave the remaining slots empty.
10. Similarly, map the physical pin 27 to the G<5>..G<8> slots.
11. To complete the process, click *OK*.

After you create the packages, the remaining steps in creating an asymmetrical parts are the same as described in the example before.

**Collapsing Expanded Scalar Pins**

You can collapse an expanded scalar pin. The steps are:

1. Right-click on the first pin of the expanded scalar pin.
2. Choose *Collapse Slots*.

**Note:** See *Creating Asymmetrical Parts using the Expand to Slots Option* on page 177 to see how to expand a scalar pin.
Creating Parts With Pins Split Across Symbols

Overview

Part Developer lets you create parts in which the pins can be split across symbols. This feature is especially helpful when you are creating a large pin count part. Two symbols for the part 24s10, one with only the logical pins and another with the logical pin A and the power pins, will be created to explain the steps to create a part with pins split across symbols. The 24s10 comes in a DIP package and has the following pin configuration:

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&lt;0..7&gt;</td>
<td>Input</td>
</tr>
<tr>
<td>CS1*</td>
<td>Input</td>
</tr>
<tr>
<td>CS2*</td>
<td>Input</td>
</tr>
<tr>
<td>Q*&lt;0..3&gt;</td>
<td>Tri State</td>
</tr>
<tr>
<td>Vcc</td>
<td>Power</td>
</tr>
<tr>
<td>GND</td>
<td>Power</td>
</tr>
</tbody>
</table>

The steps to set up the defaults, entering part properties and logical pin information and creating the packages are similar to the steps involved in creating symmetrical and asymmetrical parts. The symbol creation process is detailed below.

Note: When you split the pins across symbols, the packages also get modified appropriately.

Creating Symbols

You can create symbols in Part Developer by using the Symbol Wizard. The Wizard is a dynamic tool that enables you to create parts effectively and without errors. When you launch the wizard, it asks whether you want to split the symbols or create one symbols with all pins on it. Depending upon the response, it displays the appropriate pages.

To create symbols for the 24s10 part:
   
The Symbol Creation Wizard - Split Symbol page appears.

2. Select *Split Pins across symbols* radio button.
The Symbol Creation Wizard - Pin Selection page appears.

3. Select the following check boxes in the sym_1 column:
   - CS1*
   - CS2*
   - Q

4. Select the following check box in the sym_2 column:
   - A
   - Power Pins

5. Click Next.
The Symbol Creation Wizard - Summary page appears.

6. Click **Finish**.

This completes the creation of the two symbols for the part.

The chips.prt file is updated automatically. For example, before the symbols are created for the 24s10 part, the power pins are in the body section. However, because one of the symbols has the power pin on it, the power pins will be automatically moved to the pin section. The chips.prt file before and after symbol creation is displayed below.
The `chips.prt` File before Symbol Creation

```plaintext
OUTPUT_LOAD='(1.0,-1.0)';
OUTPUT_TYPE='(TS,TS)';
end_pin;
body
   POWER_PINS='(VCC:15;GND:16)';
   CLASS='IC';
   PART_NAME='24S10';
end_body;
end_primitive;
END.
```

The Power pins in the body section

The `chips.prt` File after Symbol Creation

```plaintext
'VCC':
   PIN_NUMBER='(15,0)';
   PIN_TYPE='POWER';
'GND':
   PIN_NUMBER='(16,0)';
   PIN_TYPE='POWER';
end_pin;
body
   CLASS='IC';
   PART_NAME='24S10';
end_body;
end_primitive;
```

The Power pins in the pin section
## Pin Types

<table>
<thead>
<tr>
<th>Pin Type</th>
<th>Function</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALOG</td>
<td>An analog pin is a passive pin. It is typically connected to a passive device. A passive device does not have a source of energy. For example, a resistor lead is a passive pin.</td>
<td>PIN_TYPE=ANALOG&lt;br&gt;NO_LOAD_CHECK=BOTH&lt;br&gt;NO_IO_CHECK=BOTH&lt;br&gt;ALLOW_CONNECT=TRUE</td>
</tr>
<tr>
<td>BIDIR</td>
<td>A BIDIR pin is an input/output pin.</td>
<td>INPUT_LOAD=(-0.01, 0.01)&lt;br&gt;OUTPUT_LOAD=(1.0, -1.0)&lt;br&gt;BIDIRECTIONAL=TRUE</td>
</tr>
<tr>
<td>INPUT</td>
<td>An input pin is one to which you apply a signal. For example, pins 1 and 2 on the 74LS00 NAND gate are input pins.</td>
<td>INPUT_LOAD=(-0.01,0.01)</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>An output pin is one to which the part applies a signal. For example, pin 3 on the 74LS00 NAND gate is an output.</td>
<td>OUTPUT_LOAD=(1.0, -1.0)</td>
</tr>
<tr>
<td>TS</td>
<td>A tri-state pin has three possible states: low, high, and high impedance. When it is in its high impedance state, a tri-state pin looks like an open circuit. For example, the 74LS373 latch has 3-state pins.</td>
<td>INPUT_LOAD=(-0.01, 0.01)&lt;br&gt;OUTPUT_LOAD=(1.0, -1.0)&lt;br&gt;OUTPUT_TYPE=(TS, TS)</td>
</tr>
</tbody>
</table>
## Pin Types

<table>
<thead>
<tr>
<th>Pin Type</th>
<th>Description</th>
<th>INPUT_LOAD</th>
<th>OUTPUT_LOAD</th>
<th>BIDIRECTIONAL</th>
<th>OUTPUT_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS_BIDIR</td>
<td>A tri-state bi-directional pin. A bidirectional pin is either an input or an output. For example, pin 2 on the 74LS245 bus transceiver is a bi-directional pin. The value at pin 1 (an input) determines the active type of pin 2, as well as others.</td>
<td>(-0.01, 0.01)</td>
<td>(1.0, -1.0)</td>
<td>TRUE</td>
<td>(TS, TS)</td>
</tr>
<tr>
<td>OC</td>
<td>An open collector gate omits the collector pull-up. Use an open collector to make &quot;wired-OR&quot; connections between the collectors of several gates and to connect with a single pull-up resistor. For example, pin 1 on the 74LS01 NAND gate is an open collector gate.</td>
<td></td>
<td>(1.0, *)</td>
<td></td>
<td>(OC, AND)</td>
</tr>
<tr>
<td>OC_BIDIR</td>
<td>An open collector bi-directional pin.</td>
<td>(-0.01, 0.01)</td>
<td>(1.0, *)</td>
<td>TRUE</td>
<td>(OC, AND)</td>
</tr>
<tr>
<td>OE</td>
<td>An open emitter gate omits the emitter pull-down. The appropriate resistance is added externally. ECL logic uses an open emitter gate and is analogous to an open collector gate. For example, the MC10100 has an open emitter gate.</td>
<td></td>
<td>(1.0, * )</td>
<td></td>
<td>(OE, OR)</td>
</tr>
<tr>
<td>OE_BIDIR</td>
<td>An open emitter bi-directional pin.</td>
<td>(-0.01, 0.01)</td>
<td>(1.0, *)</td>
<td>TRUE</td>
<td>(OE, OR)</td>
</tr>
</tbody>
</table>
### Pin Types

<table>
<thead>
<tr>
<th>Pin Type</th>
<th>Description</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER</td>
<td>A power pin expects either a supply voltage or ground. For example, on the 74LS00 NAND gate, pin 14 is VCC and pin 7 is GND.</td>
<td>PIN_TYPE=POWER&lt;br&gt;NO_LOAD_CHECK=BOTH&lt;br&gt;NO_IO_CHECK=BOTH&lt;br&gt;ALLOW_CONNECT=TRUE&lt;br&gt;Not listed on POWER_PINS line of the chips.prt file.</td>
</tr>
<tr>
<td>NC</td>
<td>A no-connect, pin-on-body pin.</td>
<td>PIN_TYPE=NC&lt;br&gt;NO_LOAD_CHECK=BOTH&lt;br&gt;NO_IO_CHECK=BOTH&lt;br&gt;ALLOW_CONNECT=TRUE&lt;br&gt;Not listed on NC_PINS line of chips.prt file</td>
</tr>
<tr>
<td>ANALOG</td>
<td>An analog pin.</td>
<td>NO_LOAD_CHECK = 'BOTH'&lt;br&gt;NO_IO_CHECK = 'BOTH'&lt;br&gt;ALLOW_CONNECT = 'TRUE'</td>
</tr>
<tr>
<td>UNSPEC</td>
<td>A pin with no specific function. This pin type is often used for connectors.</td>
<td></td>
</tr>
<tr>
<td>GROUND</td>
<td>A ground pin.</td>
<td>If the pin is in the body section&lt;br&gt;POWER_PINS = (GND:&lt;pin number&gt;);&lt;br&gt;GROUND_NETS = '&lt;pin name&gt;';&lt;br&gt;If the pin is in the pin section&lt;br&gt;'&lt;pin name&gt;':&lt;br&gt;PIN_NUMBER = ('&lt;pin_number&gt;'&lt;br&gt;PINUSE = 'GROUND';</td>
</tr>
</tbody>
</table>
Checks

You can run the following checks:

- **View Verification**
- **Instantiation and Packaging**
- **Advanced View Checks**
- **VHDL Compilation**
- **Verilog Compilation**
- **Verify with Templates**

**View Verification**

You can run the following checks:

- Symbol origin is centered.
  Checks whether the origin always lies within the symbol and the symbol is at a distance less than the maximum allowed offset from the origin.

- Tristated pins have input and output loads defined.
  Checks the presence of pin properties OUTPUT_LOAD and INPUT_LOAD for every tri-state pin. This is denoted by the property OUTPUT_TYPE =TS,TS.

- Mandatory properties present in package file.
  Checks whether the properties named BODY_NAME, PART_NAME, CLASS, and JEDEC TYPE are present in the chips.prt file.

- Consistent symbol name in symbol and package file.
  Checks whether the symbol text is the same as BODY_NAME in the chips.prt file.

- Consistent symbol and package in pin list.
  Checks whether the pins are the same across symbol and package views.
Instantiation and Packaging

Instantiation and Packaging checks include the following:

- Use Project ptf files for verification
  
  If you select this rule, part table files are used in instantiation and packaging.
  
  If no part table files are specified in the project file, the cell-level ptf is used by default.

- Use allegro board (netrev)
  
  If you select this option, the part or library is verified for the complete Front to Back flow.

- Generate Pass/Fail report
  
  This option is enabled only if you select more than one part or when you select a library that has more than one part.

  **Note:** If you select the Generate Pass/Fail option, each part is verified separately. This is a time consuming process.

Advanced View Checks

Select this option to launch CheckPlus. You can run your own custom-defined checks using CheckPlus.

VHDL Compilation

Use this option to compile the generated VHDL wrapper. You can use either NCVHDL or CV to compile the wrapper. You can specify the tool to compile the VHDL wrapper in the Enter command in the VHDL compilation dialog box. This dialog box is displayed when you click the Options button in the Verifications dialog box.

Verilog Compilation

Use this option to compile the generated Verilog wrapper. You can use NCVERILOG to compile the wrapper. You can specify the tool to compile the Verilog wrapper in the Enter command for Verilog compilation dialog box. This dialog box is displayed when you click the Options button in the Verifications dialog box.
Verify with Templates

Select this option to verify a selected part against a template. The result of the verification is displayed in a report. The verification is done as per the following rules:

Property Checks

The property check is done on all packages for the following:

- All properties listed in the template for a package must exist in each package of the part
- The value of the property in each package must match the value in the template unless the value in template is "?" or blank.

Pin Load Checks

This is done on all pins in all packages as per the following rules:

- If PINUSE="UNSPEC" exists for a pin, all checks are bypassed on that pin
- If a pin type is not determined, it is an error
- If a pin type is determined, its load value is checked against the load value of that pin type in the template. An error is generated if the load values don’t match.
- Error is shown if any of the loads for a pin type is missing

Symbol Checks

All Symbols are checked for a given part as per the following rules:

- All symbols must have at least one connection with a line stub or a bubble else an error stating that check cannot proceed is shown.
- All lines are assumed to be vertical or horizontal. Arcs are not supported in this release.
- Each Bubble is interpreted to have two virtual stub lines - horizontal and vertical.
- No two connections can have the same X, Y coordinates else error is shown to the user.
- Location of connections is derived based on the direction of the stubs attached to the connection. Therefore, only connections that have a stub or a bubble on them are checked.
- Stub length is calculated based on the integer average of all stub lengths.
The outline of the body is derived by searching for the perpendicular line from the end of stub. As the stub size varies, the search is made within the range of the stub size variance. After getting a single outline, the rest are traced as the ones connected the outline end points. The procedure is executed recursively for each detected outline.

Minimum pin spacing are calculated for all sides (Top, Left, Right, Bottom).

For pin texts, the property PIN_TEXT is used. If it is not found, pin texts are searched for within 1/3 of the average stub length from the end of the stub for each connection. Also the location (x,y) of the pin note must not be mis-aligned by more than 1/2 pin spacing.

Grid is derived by taking the highest common factor of all differences of values of X and Y on respective coordinates. Only the connection (Logic) grid is derived. The symbol grid is not derived even though the template mentions it as Symbol Grid.

All properties listed in the template for symbols must exist on each symbol in the part.

The value of the property in each symbol must match the value in the template unless the value in template is "?" or blank.

The alignment of the property must match the one specified in template.

The visibility of the property must match the one specified in template.

Pin Checks

Each pin is checked as per the following rules:

- Each pin based on the type as defined in the template must be at the location area in symbol as defined in the template for that type.

- The text size of the pin must match the size specified in the template.

- The Use Pin Names For Text is checked only if the template sets it to true.

- The text style for pin text is checked with value in template. If the style is vertical for top and bottom pins and horizontal for pins on left and right then the style is considered to be Automatic. The angles of 90 and 270 are considered equivalent and vertical and 0 and 180 are considered equivalent and horizontal.

- All pins with spacing less than the spacing specified in the template are marked as errors.
Grid Checks

Grid checks are done with the following rules:

- Conversions are done for calculating and matching the grid values for Inches, Metric and Fractional (Fractional is currently not supported in Part Developer). This is then matched with the template value.

Outline Checks

The outline checks are done with the following rules:

- All detected Outlines are checked to match the thickness specified in template.

Minimum Size Checks

- The minimum pin spacing values on the left and right is read for each symbol and verified against the minimum pin spacing left and right value stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.

- The minimum pin spacing values on the top and bottom is read for each symbol and verified against the minimum pin spacing top and bottom value stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.

- The minimum symbol height value is read for each symbol and verified against the minimum symbol height stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.

- The minimum symbol height width is read for each symbol and verified against the minimum symbol width stored in the template. An error is generated if the value in the symbol is less than that of the value stored in the template.

The output of the verification is displayed in a dialog box. The output is divided into two sections, Overview and Details.

In the Overview section, the overview of the differences are displayed. In the details section, the differences are detailed.
# Index

## Symbols

- [ ] in syntax 11
- brackets in syntax 11

## L

- Library Explorer
  - Launching 14
- Library Management Use Model 13
  - Designer 14
  - Librarian 13

## M

- Modifying Logical Pin List 65
  - Adding 66
  - Removing Pins 65
- Modifying Logical Pins
  - Renaming Pins 66
- Modifying Packages 66
  - Changing Package Properties 67
  - Modifying Physical Pin Mappings 67
- Modifying Symbols 68
  - Adding Logical Pin to all Symbols 70
  - Adding Logical Pins to Specific Symbols 71
  - Changing Associated Packages 68
  - Changing Pin Order 71
  - Changing Pin Positions 72
  - Modifying Symbol Properties 68

## P

- Part Creation Methodology 25
- Part Creation Situations 42
- Part Defaults
  - Loading 42
  - Saving 41
- Part Developer
  - Launching 15
- Part Developer Menus
  - Edit 21
  - File 19
  - Templates 23
  - Tools 22
  - View 22
- Part Developer Work Environment 18
- Part Table Editor
Part Developer User Guide

Launching 15
Part Template
Creating 91
Extraction 95
Part Templates
Using 94
Verifying against a Part Template 94
Pin Defaults
Setup 26, 29
Pin Interpretation Defaults
Setup 37
Property Values
Setup 26

S
Split Part
Methodology 63
Split Parts
Creation 179
Split Symbol Modification 72
Symbol Defaults
Setup 31

V
Verilog Wrapper
Creation 85
Verilog Wrapper/Map File 85
Deletion 88
Modification 88
Renaming 88
vertical bars in syntax 11
VHDL Wrapper/Map File
Creation 80
Deletion 84
Modification 84
Renaming 84
VHDL Wrapper/Map File Creation 77

X
XML
CLASS Property 101
Creating Parts Using XML 99
Duplicate Pins 102
JEDEC_TYPE 102
NC Pins 102