

# **Galaxy Custom Designer® IC Compiler Co-Design (ICC- CD) User Guide**

---

Version J-2014.12, December 2014

**SYNOPSYS®**

# Copyright and Proprietary Information Notice

© 2014 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

## Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

## Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

## Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.  
700 E. Middlefield Road  
Mountain View, CA 94043  
[www.synopsys.com](http://www.synopsys.com)

# Contents

---

Related Products and Trademarks . . . . .	ix
Related Publications . . . . .	ix
Conventions . . . . .	x
Customer Support . . . . .	xi

---

<b>1. Overview . . . . .</b>	<b>1</b>
Before You Start . . . . .	1
ICC Menu Commands . . . . .	2
Create FRAM . . . . .	2
Specifying Input Parameters . . . . .	3
Operation . . . . .	14
Place & Route . . . . .	15
Validate for ICC . . . . .	15
Export . . . . .	16
Save and Export . . . . .	16
Import . . . . .	16
Revert to ICC . . . . .	17
Run ICC . . . . .	17
Liberty Data Editor . . . . .	19
Opening the Liberty Data Editor Tool . . . . .	20
Viewing and Adjusting Liberty Data Parameters Using the Liberty Data Editor Tool . . . . .	20
Editing Extracted Data Using the Liberty Data Editor Tool . . . . .	22

---

<b>2. Import and Export . . . . .</b>	<b>25</b>
Importing Design Data from IC Compiler . . . . .	25
Importing Libraries . . . . .	25
IC Compiler - Custom Designer Platform Library Mapping Files . . .	30
Creating Target Libraries for IC Compiler Import . . . . .	31
Importing Individual Designs from IC Compiler . . . . .	33
Exporting Designs and Libraries to IC Compiler . . . . .	34
Exporting Libraries to IC Compiler . . . . .	34

## Contents

Creating IC Compiler Libraries for Export . . . . .	40
Exporting Individual Designs to IC Compiler . . . . .	43
<hr/>	
<b>3. Designing . . . . .</b>	<b>45</b>
Opening a Design From the Library Manager . . . . .	45
View Mapping between IC Compiler and Custom Designer . . . . .	46
Working on Schematic Designs . . . . .	47
Working on Layout Designs . . . . .	47
Closing Designs . . . . .	47
<hr/>	
<b>4. Preparing Libraries for Co-Design . . . . .</b>	<b>49</b>
Designs Originating on the Custom Designer Platform . . . . .	49
Designs Originating with the IC Compiler Tool . . . . .	51
Creating Timing Libraries for Multi-Voltage Power in Place and Route . . . . .	53
<hr/>	
<b>5. Editing an IC Compiler Design in a Custom Designer Environment. . . . .</b>	<b>55</b>
Custom Designer Tools Used in IC Compiler Co-design Flows . . . . .	55
Design Navigator . . . . .	56
Constraint Editor . . . . .	56
Check EM and Check Resistance . . . . .	56
Create Interconnect . . . . .	56
Implementing Analog Nets . . . . .	57
Safe Editing . . . . .	58
Namespace Mapping . . . . .	58
Object Mapping . . . . .	59
Adjusting Placement of Cells . . . . .	61
Drawing Routes Automatically . . . . .	63
Drawing Routes Interactively . . . . .	65
Performing IC Compiler Operations on Design Layouts . . . . .	66
Running the IC Compiler Tool using Tcl Scripts for Input . . . . .	67
Running the IC Compiler Tool using Console Commands . . . . .	68
Setting IC Compiler Bind Keys in Custom Designer . . . . .	68
Use Model Examples . . . . .	72
Creating Analog Pre-Routes Targeting a Specific Resistance Value . . . . .	72

Cleaning Up the Layout after Routing . . . . .	73
Working with Routing Objects in an ICC-CD Flow . . . . .	74
Paths . . . . .	75
Rectangles . . . . .	75
<hr/>	
<b>6. Digital Timing Shells for Analog Blocks . . . . .</b>	<b>77</b>
Timing Shells . . . . .	78
Creating Timing Shells for Analog Blocks . . . . .	80
Prepare to Create a Timing Shell. . . . .	80
Create Connection Definitions for the Timing Shell . . . . .	82
Define the IC Compiler Black Box . . . . .	83
<hr/>	
<b>7. Developing Digital Blocks with Co-Design . . . . .</b>	<b>89</b>
Adding Floorplanning Conditions for Digital Blocks . . . . .	90
Implementing Digital Blocks. . . . .	90
Running IC Compiler Place and Route from Design Schematics or Verilog Netlists . . . . .	91
Tracking Progress and Viewing Output Files for IC Compiler Operations	95
Designing with Place and Route Results . . . . .	96
Customizing the IC Compiler Place and Route Command File . . . . .	97
<hr/>	
<b>8. Tutorial: Constraint Rules Mapping . . . . .</b>	<b>99</b>
Before You Start. . . . .	99
Files . . . . .	100
Product Versions . . . . .	100
Task 1: Analyze the Design Using IC Compiler . . . . .	100
Task 2: Add and Open IC Compiler Libraries for this Design in Custom Designer . . . . .	104
Task 3: Open the Design Layout . . . . .	106
Task 4: Review NDRs with the Custom Designer Layout Editor . . . . .	107
Task 5: Edit and Create NDRs and Shield Constraints . . . . .	110
Task 6: Evaluate Edits with the IC Compiler Tool. . . . .	118
Task 7: Edit Routing Rules in IC Compiler. . . . .	122
Summary . . . . .	124

## Contents

Limitations .....	124
-------------------	-----

---

<b>A. Setting the User Environment. ....</b>	<b>127</b>
Tech Mapping between the Custom Designer Platform and the IC Compiler Tool. ....	127
Custom Designer Layer Purpose Pair (LPP) to IC Compiler Layer: DataType Mapping .....	127
Custom Designer Platform to IC Compiler Display Attribute Mapping . . .	129
Export to the IC Compiler Tool. ....	131
Scenario 1: Exporting without Available IC Compiler Technology or Design Data .....	131
Generate an IC Compiler Technology File .....	131
Generate IC Compiler Tech File and Library .....	131
Scenario 2: Exporting to an Existing IC Compiler Library with Matching Layers. ....	133
Scenario 3: Exporting to an Existing IC Compiler Library with Different Layer Names .....	134
Scenario 4: Exporting to an Existing IC Compiler Library with No Matching Layers. ....	137
Using the Automated Method of Exporting to IC Compiler. ....	138
Using the Layer Mapping File. ....	140
Import from the IC Compiler Tool. ....	142
Scenario 1: Importing without Available Custom Designer Technology or Design Data .....	142
Generate a Custom Designer Technology File. ....	142
Generate Custom Designer Technology File and Library .....	142
Scenario 2: Importing to an Existing Custom Designer Library with Matching Layers. ....	144
Scenario 3: Importing to an Existing Custom Designer Library with Different Layer Names .....	146
Scenario 4: Importing to an Existing Custom Designer Library with No Matching Layers. ....	148
Using the Automated Behavior of Import From IC Compiler .....	149
Using the Layer Mapping File. ....	151
Scenario 5: Importing to an Existing Custom Designer Library with Read-Only Technology .....	153
Pre-Route Flow .....	156
Creating Guard Ring Definitions for an IC Compiler Design in Custom Designer. ....	156
Using Separate OpenAccess Technology for an Opened IC Compiler Library .....	160

## Contents

Setup Using LEF Files .....	162
Scenario 1: Importing LEF Files to Create an IC Compiler Library .....	162
Scenario 2: Importing LEF files to create a Custom Designer library ...	163

---

<b>Index</b> .....	165
--------------------	-----

## Contents



# About This Manual

---

This manual is a companion to the *Custom Designer Platform Guide*, describing how to use the IC Compiler tool in conjunction with the Custom Designer platform for co-design. You can

- design analog nets in a digital context, performing a pre-routing function before place and route.
- run place and route.
- bring the design from the IC Compiler tool to the Custom Designer platform at any point during floorplanning, placement, or power/ground route drawing and clock tree synthesis.
- perform IC Compiler operations directly from the Custom Designer platform.

---

## Related Products and Trademarks

This manual refers to the following products:

Synopsys IC Compiler™  
Synopsys SolvNet® support site

---

## Related Publications

Additional information about IC Compiler Co-Design with Custom Designer is available on SolvNet.

For additional information about Custom Designer, see

- The documentation installed with the Custom Designer software and available through the Custom Designer Help menu
- The Custom Designer Release Notes, available on SolvNet (see [Accessing SolvNet on page xi](#))
- Documentation on the Web, which provides HTML and PDF documents and is available on SolvNet (see [Accessing SolvNet on page xi](#))

---

## Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates command syntax.
<i>Italic</i>	Indicates a user-defined value, such as <i>object_name</i> .
Purple	<ul style="list-style-type: none"> <li>Within an example, indicates information of special interest.</li> <li>Within a command-syntax section, indicates a default value, such as:  <pre>include_enclosing = true   false</pre> </li> </ul>
<b>Bold</b>	<ul style="list-style-type: none"> <li>Within syntax and examples, indicates user input—text you type verbatim.</li> <li>Indicates a graphical user interface (GUI) element that has an action associated with it.</li> </ul>
[ ]	Denotes optional parameters, such as: <pre>write_file [-f filename]</pre>
...	Indicates that parameters can be repeated as many times as necessary: <pre>pin1 pin2 ... pinN</pre>
	Indicates a choice among alternatives, such as <pre>low   medium   high</pre>
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
<b>Edit &gt; Copy</b>	Indicates a path to a menu command, such as opening the <b>Edit</b> menu and choosing <b>Copy</b> .
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing the C key.

---

---

## Customer Support

Customer support is available through the Synopsys SolvNet customer support website and by contacting the Synopsys support center.

---

### Accessing SolvNet

The SolvNet support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNet site:

1. Go to the web page at <https://solvnet.synopsys.com>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

If you need help using the site, click **Help** on the menu bar.

---

### Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact Synopsys support in the following ways:

- Go to the Synopsys [Global Support](#) site on [synopsys.com](http://synopsys.com). There you can find e-mail addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNet site or the Synopsys Global Support site and [open a case online](#) (Synopsys user name and password required).

Customer Support

## Overview

---

*This chapter provides a general overview of the menu items in the Custom Designer ICC menu.*

This chapter describes the following topics:

- [Before You Start](#)
- [ICC Menu Commands](#)

---

## Before You Start

Add the ICC menu to the design menu:

- In any design window, choose **Tools > ICC Link** to add the ICC menu choice to the design menu, shown in [Figure 1](#).

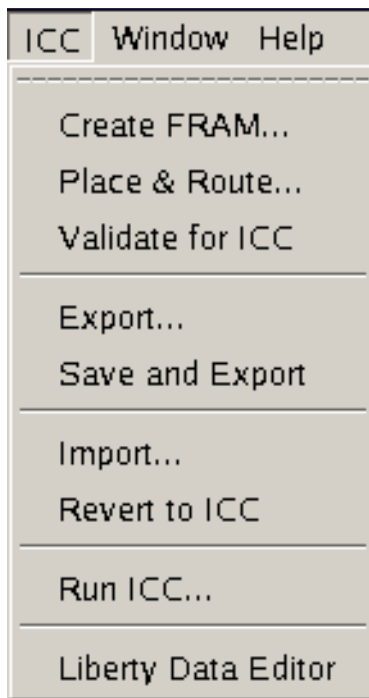


Figure 1 ICC Menu

---

## ICC Menu Commands

The ICC menu commands are described in the following sections.

---

### Create FRAM

Use the following procedure to generate ICC FRAM views:

1. Choose **ICC > Create FRAM**.

The **Create ICC FRAM** dialog box opens.

**Note:** A different version of the dialog opens depending on the library type, OA or ICC.

2. Specify the input and layer blockage parameters, as described in the [Specifying Input Parameters](#) section.

3. You can also create an ICC FRAM view using the Tcl command [xt::CreateICCFRAME](#).
4. Click **OK** or **Apply** to start the operation.

## Specifying Input Parameters

Tcl Command: <a href="#">xt::showCreateICCFRAME</a>
---

The **Create ICC FRAME** dialog box has several tabs. The following procedures explain how to complete the fields on each tab.

- [Main Tab \(Required Information\)](#)
- [Layer Blockage Tab](#)
- [PR Boundary Tab \(Optional\)](#)
- [Export Options Tab](#)
- [Script Tab](#)

**Main Tab (Required Information)**

Figure 2 shows the **Main** tab of the **Create ICC FRAM** dialog box.

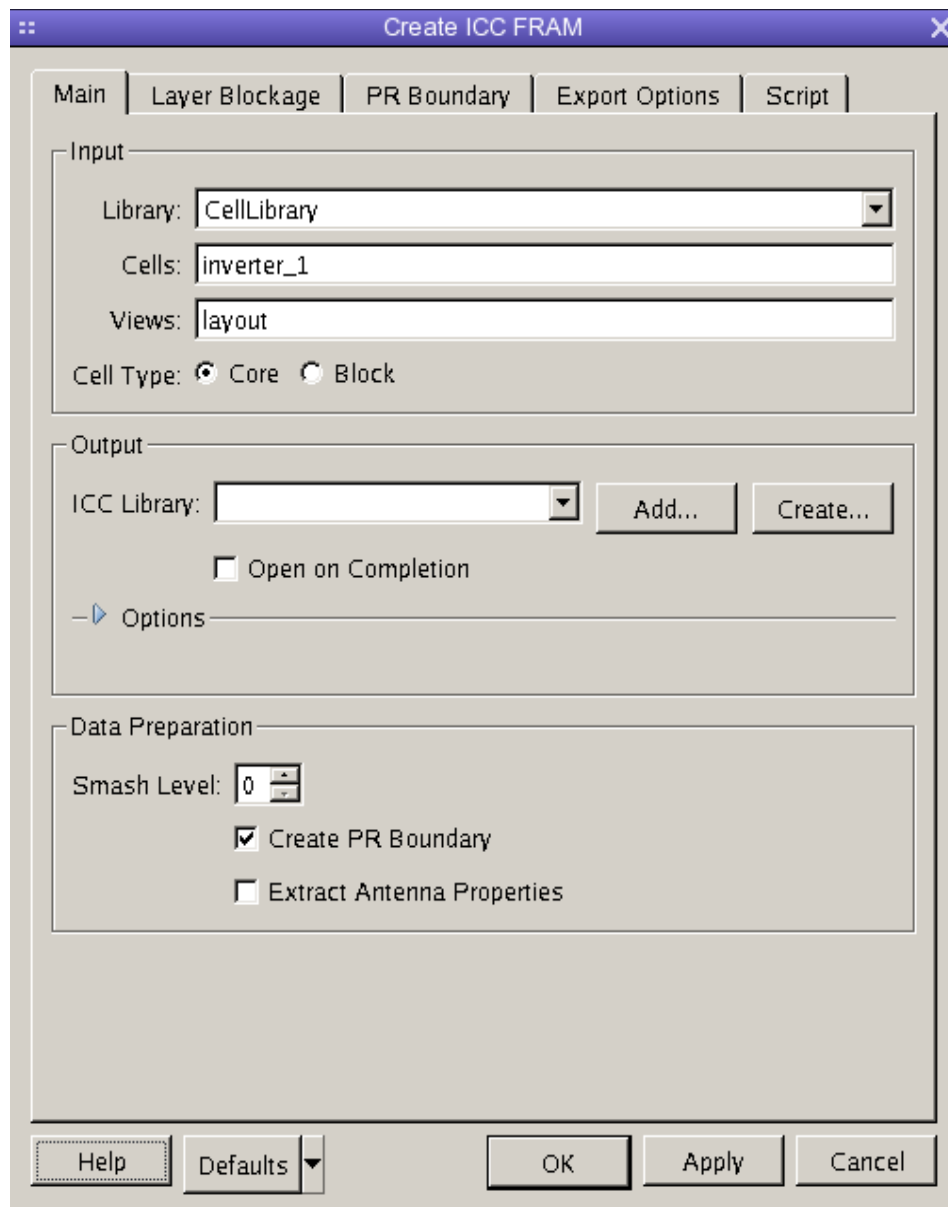


Figure 2 Create ICC FRAM Dialog: Main Tab



Use the following procedure to specify the required inputs on the **Main** tab of the dialog box.

1. In the **Input** area of the dialog box, specify the source cellView for which to create a FRAM view.

When first opened, the tool populates the dialog box with information about the edit cellView in the active Layout Editor window.

2. In the **Output** area of the dialog box, specify the target library to store the FRAM views by doing one of the following:

- If you do not have an existing target library, accept the default Library:

`<workspace>/icclibs/<source_library_name>`

where:

*workspace* is your current design directory.

*source\_library\_name* is the same name as the Top Library name in the Input region of the dialog box.

- If you have an existing target library, browse to select it.

**Note:** If the input cellviews are in an ICC library, the **Output** area contains only the option to open the FRAM view in ICC when the operation finishes.

(Optional) Specify whether to open the FRAM view in ICC when the operation finishes.

(Optional) Specify the following additional options:

- Whether to overwrite any existing cells
- Whether to create an embedded netlist
- Whether to create layers
- Whether to remaster the instances from CEL to FRAM

3. (Optional) In the **Data Preparation** region of the dialog box, you can specify the following:

- **Smash Level**

This option lets you control processing access to geometries inside the hierarchy of a layout cell. You can set a level that either allows or excludes access.

- **Create PR Boundary**

When selected, the **PR Boundary tab** and options are available. See [PR Boundary Tab \(Optional\) on page 10](#).

**Note:** This field is only available when **Cell Type: Core** is selected in the **Input** region of the dialog box.

- **Extract Antenna Properties**

**Note:** The Hercules or IC Validator executable is required to use the “Extract Antenna Properties” option.

See also

[Specifying Input Parameters on page 3](#)  
[Layer Blockage Tab on page 6](#)  
[PR Boundary Tab \(Optional\) on page 10](#)  
[Export Options Tab on page 12](#)  
[Script Tab on page 13](#)  
[Operation on page 14](#)

***Layer Blockage Tab***

The **Layer Blockage** tab shows all the poly and metal layers in the current technology. The tab content depends the input library selection.

Figure 3 shows the **Layer Blockage** tab of the dialog box for **Cell Type: Core**.

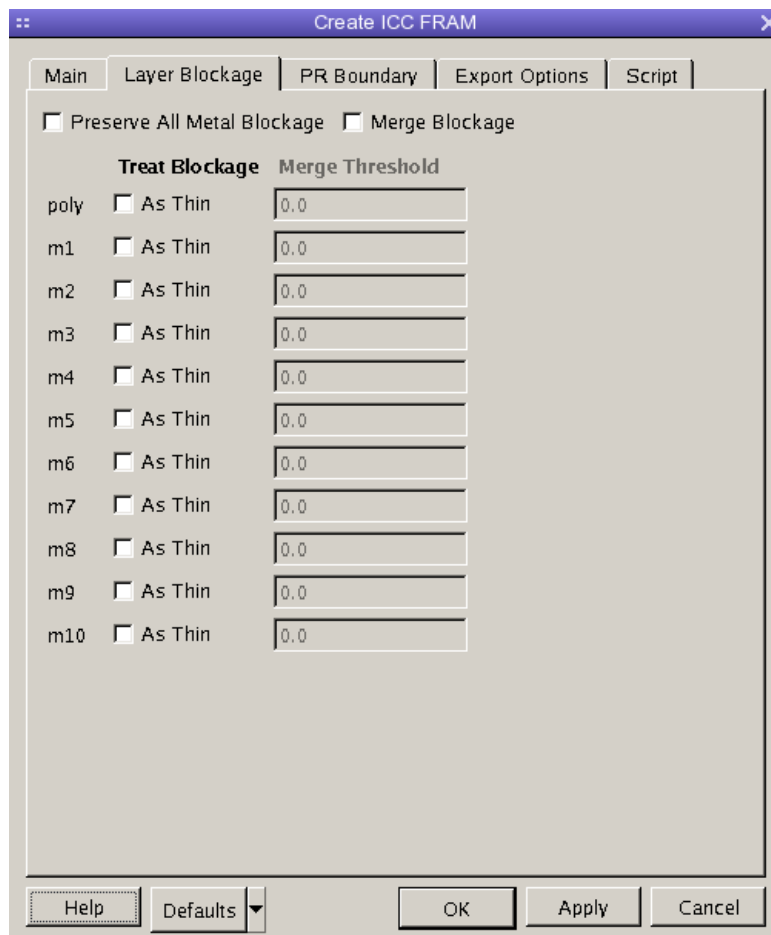


Figure 3 Create ICC FRAM Dialog: Layer Blockage Tab (Cell Type: Core)

Figure 4 shows the **Layer Blockage** tab of the dialog box for **Cell Type: Block**.

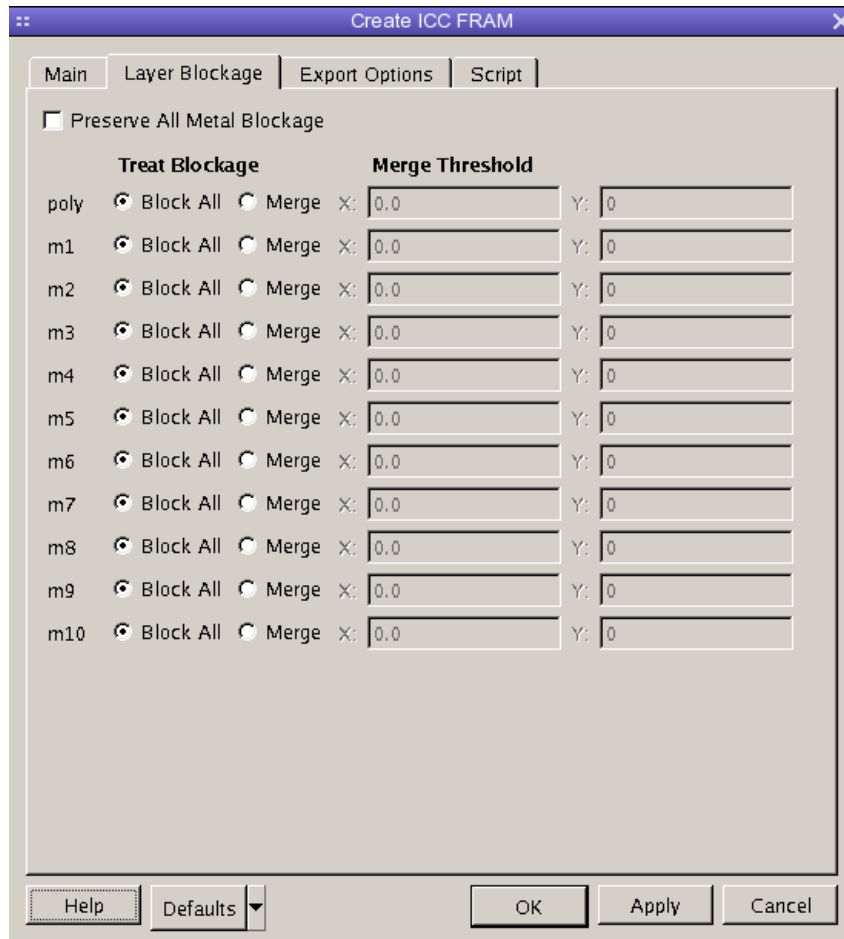


Figure 4 Create ICC FRAM Dialog: Layer Blockage Tab (Cell Type: Block)

There are two methods for specifying the layer blockages:

- Enable the **Preserve All Metal Blockages** option.  
When selected, all metal blockages retain their original shapes. They will not be trimmed or cut to permit pin access.
- Manually configure individual layers.  
For each layer, you can specify the following:

Setting	Description
Treat Blockage	<p>Specifies how to treat blockage layers:</p> <ul style="list-style-type: none"> <li>▪ <b>As Thin:</b> Treats the blockage layer as thin. Thin wires are wires/blockages that have the minWidth value. By default, real metal blockages are treated as fat wire if fat or thin wire if thin. It causes real metal blockages to be treated as thin wire for design rule checking purposes.</li> <li>▪ <b>Block All:</b> Creates a blockage for the specified layer. Select this option when you want the FRAM view to contain few areas for over-the-cell routing.</li> <li>▪ <b>Merge:</b> Merges nearby shapes on the specific layer according to the threshold values you specify.</li> </ul>
Merge Threshold	<p>Specifies how the blockage shapes are merged depending on the Cell Type set in the Main tab:</p> <ul style="list-style-type: none"> <li>▪ <i>Core Cell Type:</i> Merges threshold with a single value.</li> <li>▪ <i>Block Cell Type:</i> Specifies the X and Y directions for merging nearby shapes. When the X and Y values are both 0, the original blockage shape is retained. If you specify a large value, nearby shapes will merge into a larger blockage.</li> </ul>

See also

[Specifying Input Parameters on page 3](#)  
[Main Tab \(Required Information\) on page 4](#)  
[PR Boundary Tab \(Optional\) on page 10](#)  
[Export Options Tab on page 12](#)  
[Script Tab on page 13](#)  
[Operation on page 14](#)

### ***PR Boundary Tab (Optional)***

Figure 5 shows the **PR Boundary** tab of the dialog box.

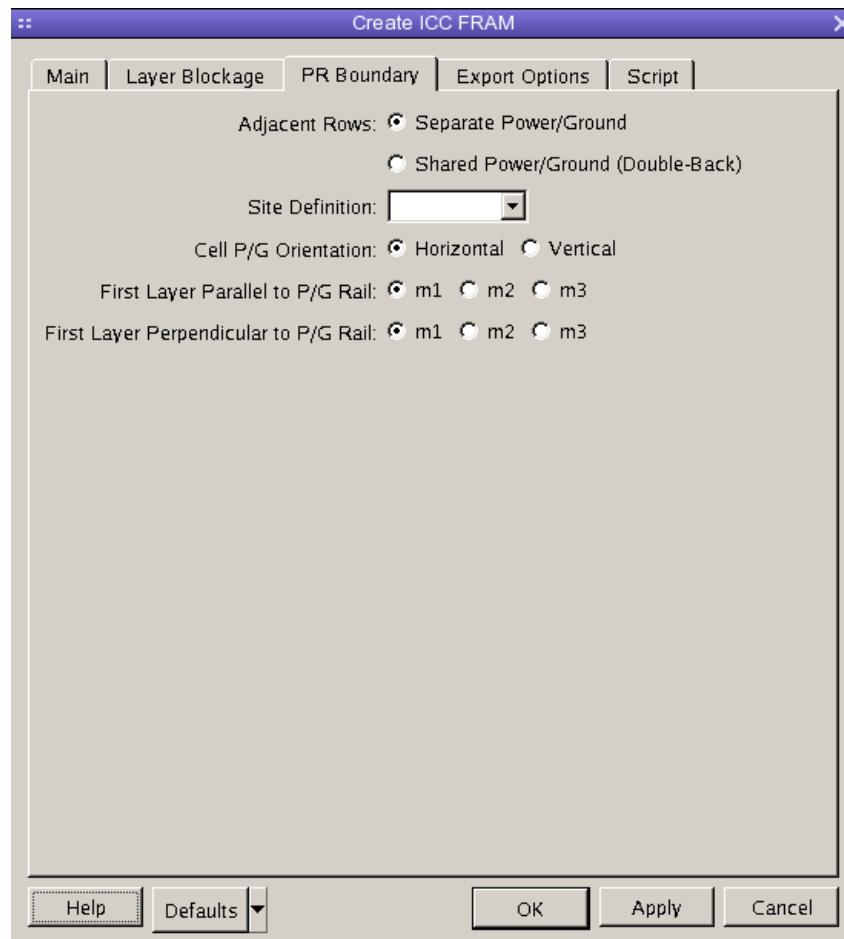


Figure 5 Create ICC FRAM Dialog: PR Boundary Tab

The options on the **PR Boundary** tab are available only when you enable the **Create PR Boundary** option on the **Main** tab.

**Note:** This field is only available when **Cell Type: Core** is selected in the **Input** region on the **Main** tab.

[Table 1](#) describes the options. For more information, see the *Milkyway Environment* online help.

*Table 1 PR Boundary Options*

Option	Description
Adjacent Rows	Sets the height of the PR boundary based on a cell placement: <ul style="list-style-type: none"> <li>▪ <i>Separate</i>: The cells do not overlap.</li> <li>▪ <i>Shared</i>: The cell rows are paired, with one row in each pair flipped and the cells placed so that the bottom power and ground rails of the flipped cells overlap with the top power and ground rail of the unflipped cells.</li> </ul>
Site Definition	A list of the sites (unit tiles) where you can place cells in a row.
Cell P/G Orientation	Specify the orientation — horizontal or vertical — for the power and ground rails.
First Level Parallel to P/G Rail	Select the first metal layer that you want to be parallel to the power and ground rail.
First Level Perpendicular to P/G Rail	Select the first metal layer that you want to be perpendicular to the power and ground rail.

#### See also

[Specifying Input Parameters on page 3](#)  
[Main Tab \(Required Information\) on page 4](#)  
[Layer Blockage Tab on page 6](#)  
[Export Options Tab on page 12](#)  
[Script Tab on page 13](#)  
[Operation on page 14](#)

### Export Options Tab

Figure 6 shows the **Export Options** tab of the dialog box.

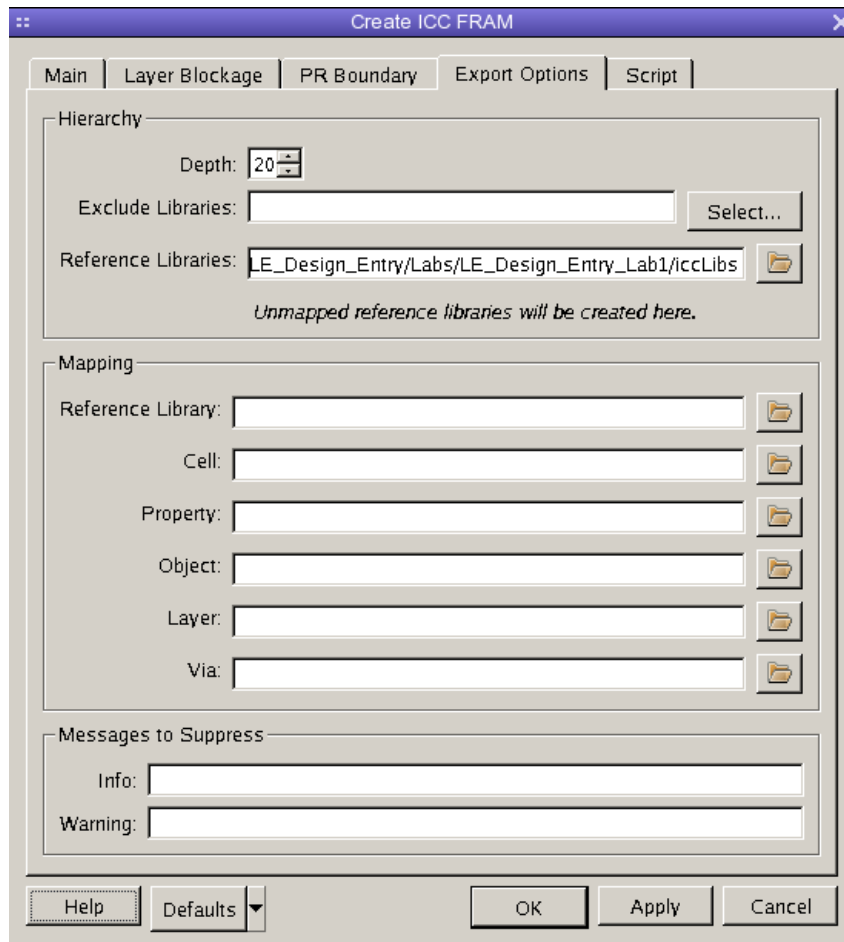


Figure 6 Create ICC FRAM Dialog: Export Options Tab

The **Export Options** tab specifies the options used for export.

- In the **Hierarchy** region of the dialog box, specify the hierarchy **Depth**, **Exclude Libraries**, and the **Reference Libraries** location where unmapped reference libraries are created. If the depth is set to 0, the libraries options are disabled.
- In the **Mapping** region of the dialog box, specify optional mapping files for **Reference Library**, **Cell**, **Property**, **Object**, **Layer**, and **Via**.
- In the **Messages to Suppress** region of the dialog box, specify the **Info** and **Warning** messages to suppress entered as space-delimited lists of message IDs as reported by the translator.



See also

[Specifying Input Parameters on page 3](#)  
[Main Tab \(Required Information\) on page 4](#)  
[Layer Blockage Tab on page 6](#)  
[PR Boundary Tab \(Optional\) on page 10](#)  
[Script Tab on page 13](#)  
[Operation on page 14](#)

### Script Tab

Figure 7 shows the **Script** tab of the dialog box.

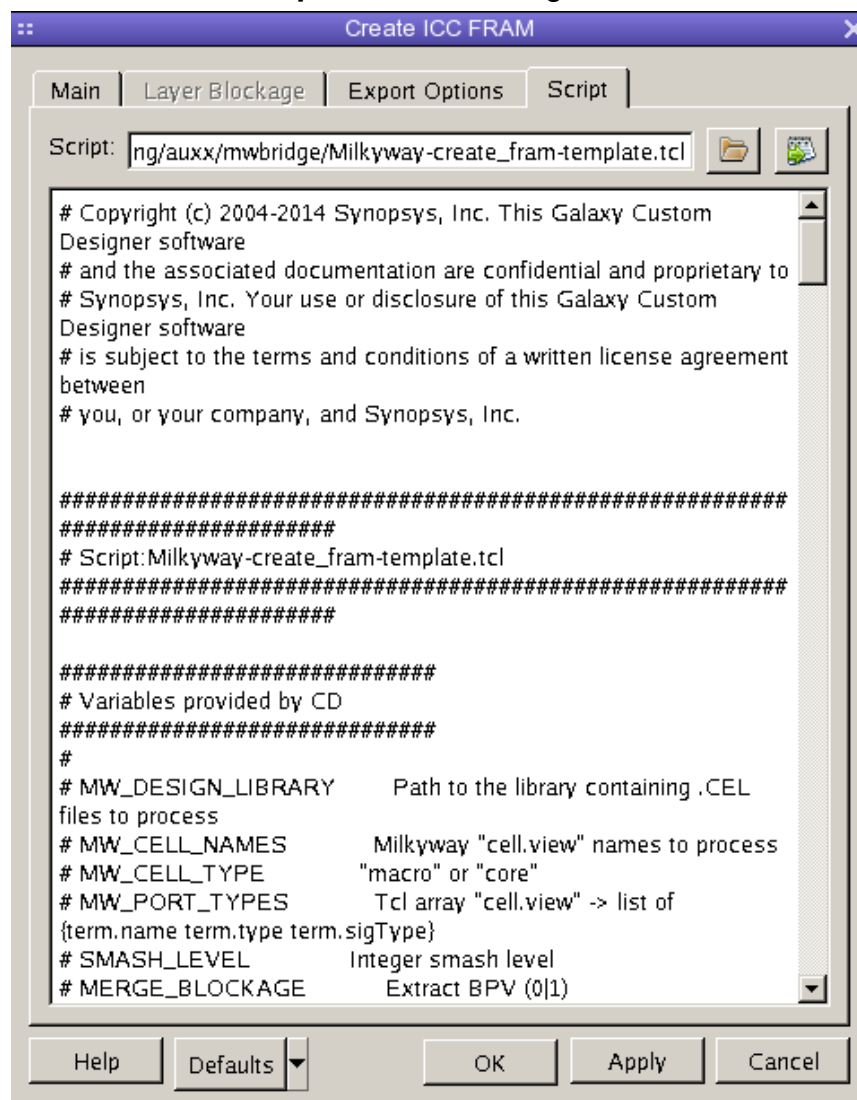


Figure 7 Create ICC FRAM Dialog: Script Tab

**Note:** The **Script** tab displays a template script only. You can copy this template and add your own code to customize the way FRAM views are created. The tab does not generate a script based on input in the dialog box.

See also

[Specifying Input Parameters on page 3](#)  
[Main Tab \(Required Information\) on page 4](#)  
[Layer Blockage Tab on page 6](#)  
[PR Boundary Tab \(Optional\) on page 10](#)  
[Export Options Tab on page 12](#)  
[Operation on page 14](#)

## Operation

Prior to performing the FRAM creation operation, the tool performs the following checks to ensure that the external applications can be found in your PATH:

- The Milkyway executable  
If this executable cannot be found, the Locate Milkyway Executable dialog box prompts you to specify the path to the executable. After entering a valid path, clicking OK adds the specified path to your PATH variable. Pressing Cancel aborts the operation.
- The `icc_shell` executable is required if the Open on Completion option is enabled and the `leCreateFRAMViewer` preference is set to ICC.
- The Hercules or IC Validator executable is required to use the “Extract Antenna Properties” option.

When you click **OK** or **Apply** in the Create Milkyway ICC FRAM dialog box, the Bridge:

1. Creates a script that performs the FRAM view creation.
2. Saves the current state to the `prefs.xml` file within the `iccbridge` directory.
3. Exports the specified designs to ICC.  
Cells are created according to the output specified in the Create FRAM dialog box.
4. Generates SMASH and FRAM views for each converted design.
5. (Optional) Opens all resulting FRAM views with ICC.

See also

[Specifying Input Parameters on page 3](#)  
[Main Tab \(Required Information\) on page 4](#)  
[Layer Blockage Tab on page 6](#)  
[PR Boundary Tab \(Optional\) on page 10](#)  
[Export Options Tab on page 12](#)  
[Script Tab on page 13](#)

---

## Place & Route

See [Running IC Compiler Place and Route from Design Schematics or Verilog Netlists on page 91](#).

---

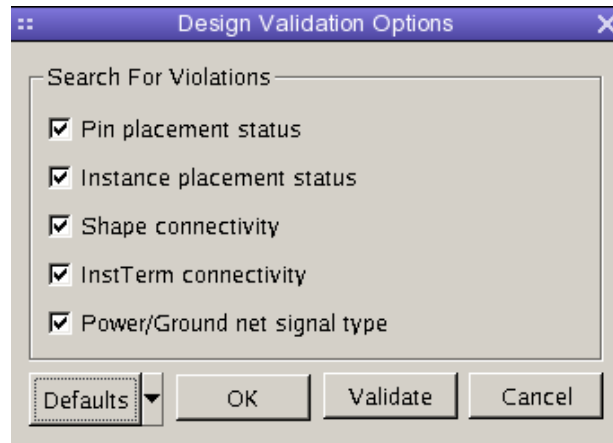
## Validate for ICC

Many design requirements need to be added in Custom Designer to export a block as an ICC-ready CEL using **Export To ICC** with **Create Embedded Netlist** enabled. For this purpose, **Validate for ICC** checks whether the design to be exported satisfies some minimum conditions, as follows:

- All the pins should have a placement status.
- All the shapes should have connectivity.
- All the instances' instTerms have connectivity.
- All the instances within the boundary have a placement status.
- All the instances outside the boundary does not require a placement status.
- Correct sigType for power/ground nets.

If there are any errors, they will be marked by error markers. Use the Marker Browser assistant to browse and fix the errors.

Choose **ICC > Validate for ICC** to open the **Design Validation Options** dialog.



*Figure 8 Design Validation Options Dialog*

Choose to search for the following design violations:

- **Pin placement status**
- **Instance placement status**
- **Shape connectivity**
- **InstTerm connectivity**
- **Power/Ground net signal type**

---

## Export

See [Exporting Designs and Libraries to IC Compiler on page 34](#).

---

## Save and Export

See [Exporting Designs and Libraries to IC Compiler on page 34](#).

---

## Import

See [Importing Design Data from IC Compiler on page 25](#).

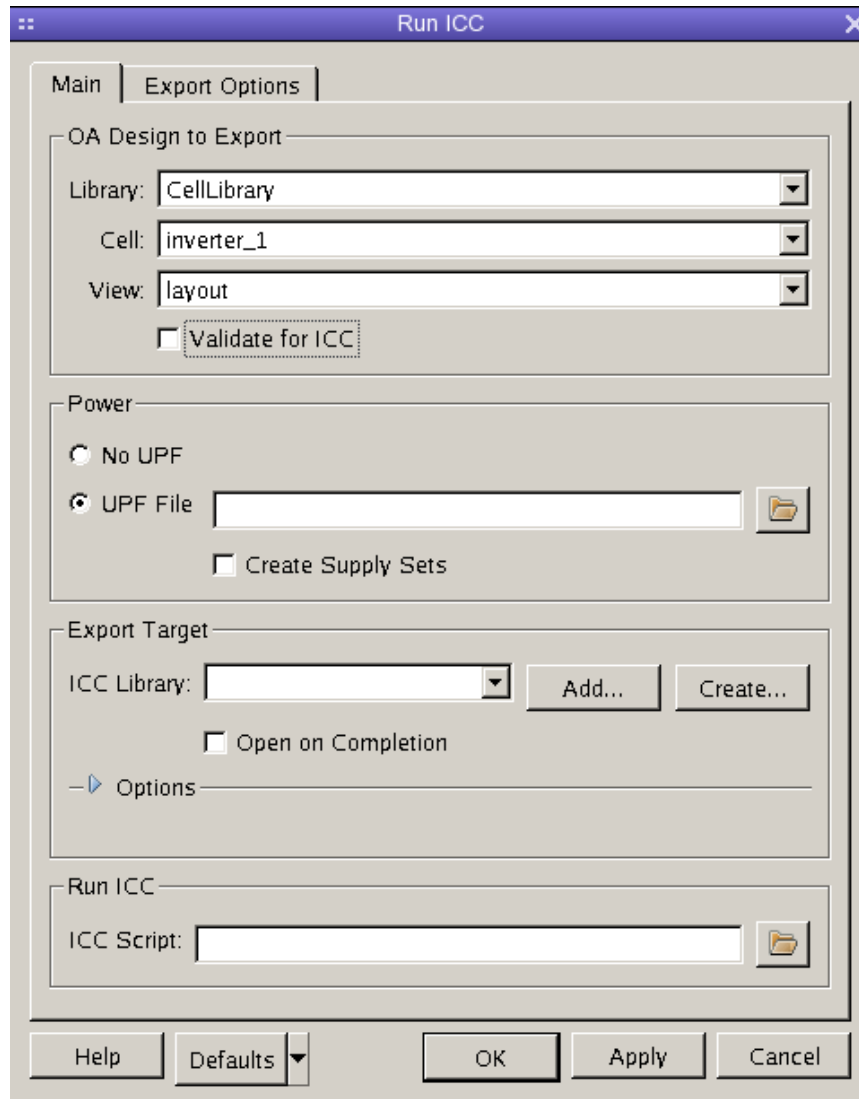
## Revert to ICC

Choose **ICC > Revert to ICC** to replace the edit design with a copy imported from IC Compiler. **ICC > Revert to ICC** opens the **Import from ICC** dialog with the **Input** region of the dialog set to the specified design and disabled. See [Importing Libraries on page 25](#) for more information.

---

## Run ICC

Choose **ICC > Run ICC** to open the **Run ICC** dialog. See [Performing IC Compiler Operations on Design Layouts on page 66](#) for more information.



*Figure 9 Run ICC Dialog: Main Tab*

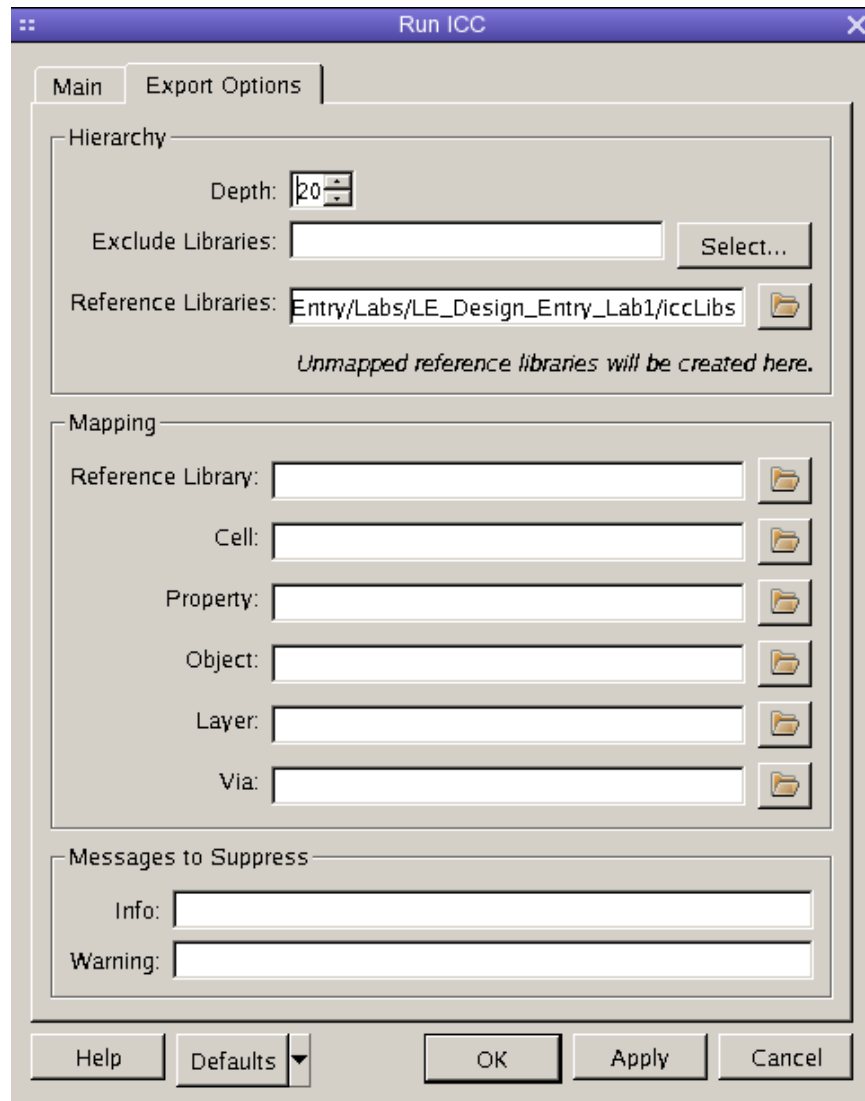


Figure 10 Run ICC Dialog: Export Options Tab

---

## Liberty Data Editor

You can use the Liberty Data Editor to view and define settings and values for a Liberty library file. The tool generates the file as specified. To review the file, open it in a text editor.

To export the Liberty Library file once it has been generated, see [Exporting Design Data in Liberty Library File Format on page 74](#).

These sections cover:

- [Opening the Liberty Data Editor Tool on page 20](#)
- [Viewing and Adjusting Liberty Data Parameters Using the Liberty Data Editor Tool on page 20](#)
- [Editing Extracted Data Using the Liberty Data Editor Tool on page 22](#)

## **Opening the Liberty Data Editor Tool**

To open the Liberty Data Editor, be sure ICC is activated. Then:

- In any design window, choose **ICC > Show Liberty Data Editor**.

See next

[Viewing and Adjusting Liberty Data Parameters Using the Liberty Data Editor Tool on page 20](#)

More about using the [Liberty Data Editor on page 19](#)

## **Viewing and Adjusting Liberty Data Parameters Using the Liberty Data Editor Tool**

You can view and adjust settings in the **Main** tab of the **Edit Liberty Data** dialog ([Figure 11](#)).



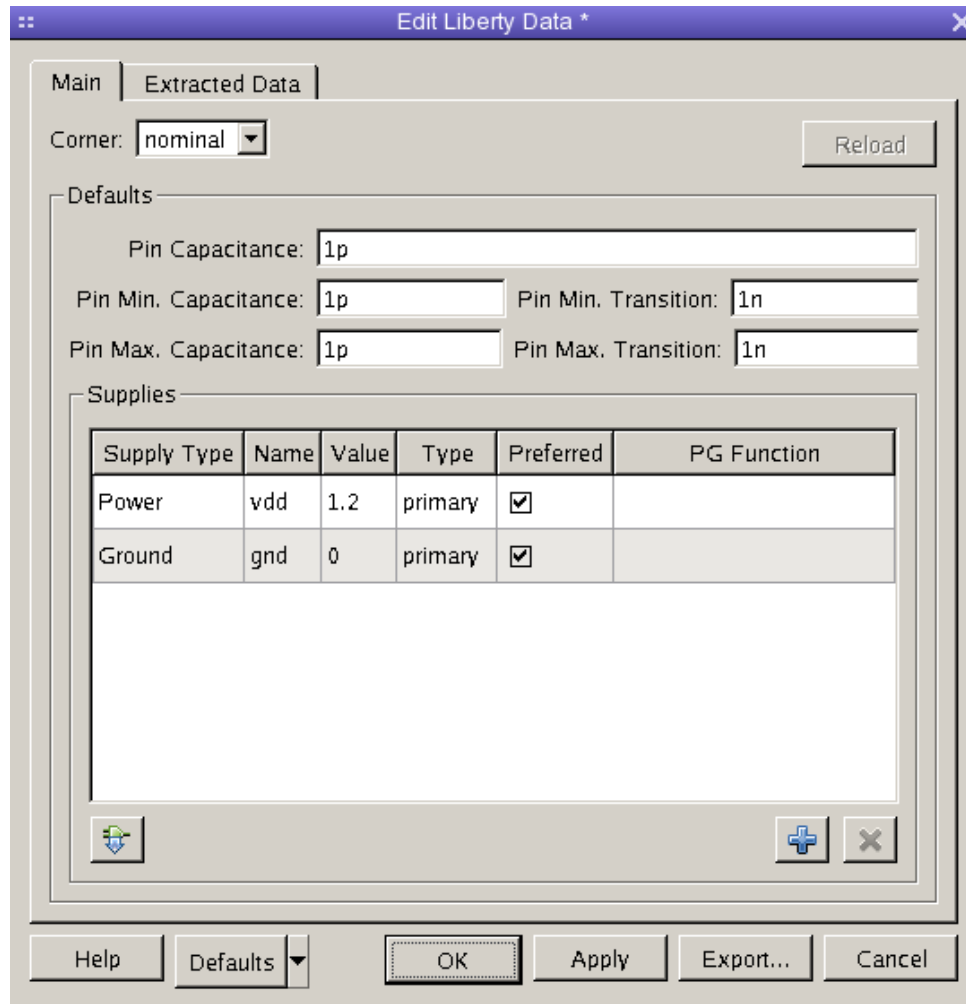



Figure 11 Editor Liberty Data: Main Tab

To view and adjust settings:

1. Begin by [Opening the Liberty Data Editor Tool on page 20](#).
2. In the **Edit Liberty Data** dialog, the **Corner** drop-down menu shows corners specified in the current design. This does not allow you to specify a corner name. If there is no corner in the current design, the corner "nominal" is created automatically. Its data is extracted from the pins of the Pins Source View and from the capacitance information of the Capacitances Source View in the **Extracted Data** tab.  
Multiple corners can be created for the same design using the command `db::copyLibertyData`.

3. Click the **Reload** button to reload the corner data to revert changes. This is only enabled when there is stored corner data in the design.
4. Examine the **Defaults** section, which contains default pin capacitance (used when capacitance extraction was not successful) and the **Supplies** table. The loaded (or extracted) Liberty data is shown in the **Supplies** table and in the **Pin Capacitance** table of the **Extracted Data** tab.  
The default pin capacitance is also used when **Extracted Data: Capacitances Source View** is empty, or when there is no capacitance information in this source for the given pin.
5. Notice the **Supplies** table section contains a button for copying supply pins  and buttons for adding and deleting supplies, **+** and **X**.  
**Copy Supply Pins** optionally copies power supply pins from a cellView you specify in the **Copy Supply Pins** dialog box, along with **View Search List** and **View Stop List**.
6. **OK**, **Apply**, and **Export** buttons. The Export button launches the **Export Liberty File** dialog (described in [Exporting Design Data in Liberty Library File Format on page 74](#)).

See also

[Editing Extracted Data Using the Liberty Data Editor Tool on page 22](#)  
[More about using the Liberty Data Editor on page 19](#)

## **Editing Extracted Data Using the Liberty Data Editor Tool**

You can edit extracted data from the **Extracted Data** tab.

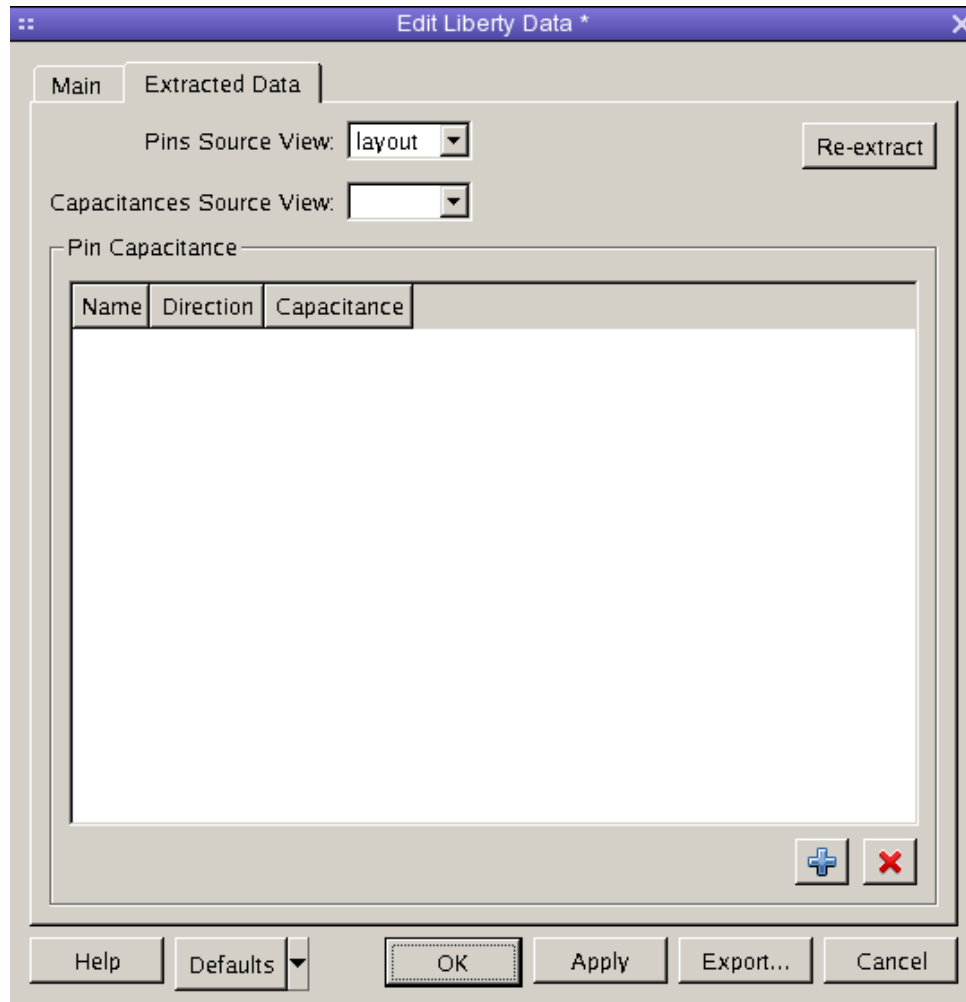


Figure 12 Editor Liberty Data: Extracted Data Tab

This tab shows the pins and capacitances source view names, as well as the **Pin Capacitance** data table.

- **Pins Source View** is the current view name by default.
- **Capacitance Source View** is empty by default.
- The **Re-extract** button re-extracts the Liberty data from the desired pins and/or capacitances' source cell. To extract pins only, set the capacitance cell value to empty. To extract capacitances only, set the pins cell value to

empty. To extract both pins and capacitances, set the corresponding table cell values to the desired cell views. After re-extracting, the current data in the **Pins Capacitance** table is replaced.

- The **Pin Capacitance** table is populated with Liberty data extracted from these cell views if the corner is created automatically. If the corner pre-exists, the values of pins and capacitances source views are set from the stored liberty information associated with the corner.

The pin names in the **Pin Capacitance** table are editable. You can also add a new pin using the **+** button, or delete a selected pin using the **X** button.

To show/hide columns in the **Pin Capacitance** table, right-click in the table header to access the **Show/Hide Columns** dialog. The following columns are available:

- **Name**
- **Direction**
- **Capacitance**
- **Min. Capacitance**
- **Max. Capacitance**
- **Min. Transition**
- **Max. Transition**
- **Related Power**
- **Related Value**
- **Function**

Double-click on the **+** button to the left of a pin name in the **Extracted Data** tab to open the **Bus Bits** tab.

See also

More about using the [Liberty Data Editor on page 19](#)

[Exporting Design Data in Liberty Library File Format on page 74](#)

## Import and Export

---

*Importing and exporting design libraries and individual designs from IC Compiler.*

This chapter covers:

- [Importing Design Data from IC Compiler on page 25](#)
- [Exporting Designs and Libraries to IC Compiler on page 34](#)

---

### Importing Design Data from IC Compiler

These sections cover:

- [Importing Libraries on page 25](#)
- [Importing Individual Designs from IC Compiler on page 33](#)

---

#### Importing Libraries

Import libraries from the IC Compiler tool when you want to design using both the Custom Designer platform and the IC Compiler tool.

When you import IC Compiler design data, a bridge process between the two tools takes the source library containing the data you want to import, converts it for Custom Designer tool access, and adds it to a target library for your use.

## Chapter 2: Import and Export

### Importing Design Data from IC Compiler

To import a design library from IC Compiler:

1. (Optional) If there is no designated Custom Designer platform target library to receive imported data, begin by [Creating Target Libraries for IC Compiler Import on page 31](#).
2. From the Console, choose **File > Import > From ICC**. The **Import from ICC** dialog comes up ([Figure 13](#)).

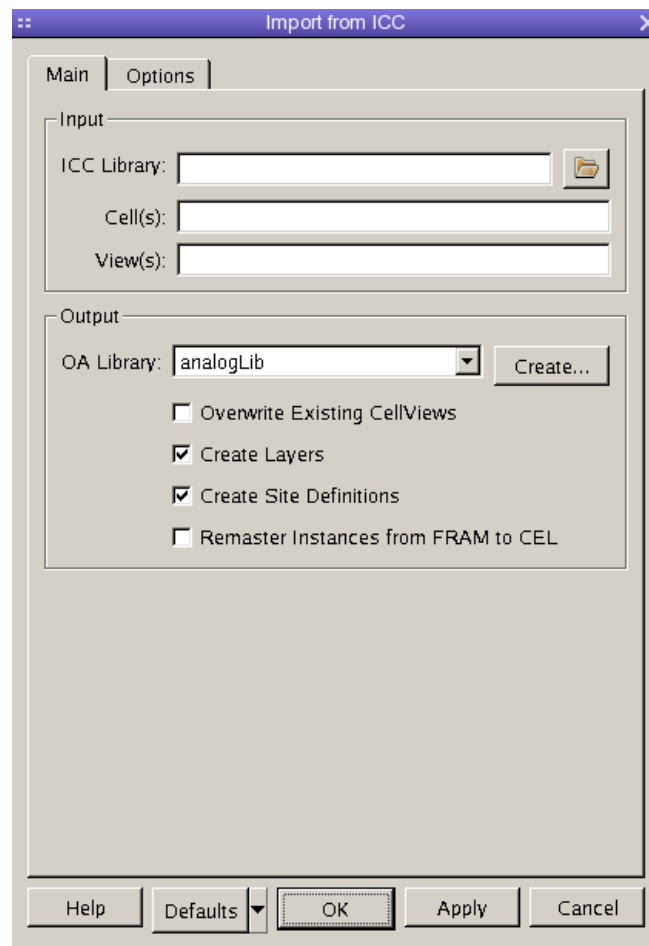


Figure 13 Import from ICC Dialog: Main Tab

3. In the **Main** tab (shown in [Figure 13](#)), specify required input details:
  - a. In the **ICC Library** field, enter the path to a valid IC Compiler library. This is the source file to be converted.
  - b. Enter a space-separated list of cells. If empty, all cells are converted.

- c. Enter a space-separated list of views. If empty, all views are converted.
  - d. In the **Output** area, from the **OA Library** menu, select a target library to receive converted data.

The menu displays the libraries listed in your `lib.defs` file, if the file exists. If the menu is empty, or if you want to create a new library, click **Create** and follow the procedure for [Creating Target Libraries for IC Compiler Import](#).
  - e. Use the check boxes to choose whether or not to **Overwrite Existing CellViews**, **Create Layers**, or **Create Site Definitions**.

Check **Create Layers** to import shapes on layers not referenced by the technology. If unset, these shapes will be skipped.

Check **Create Site Definitions** to create site definitions in the target technology during data translation.
  - f. (Optional) To have the detailed layout in Custom Designer, specify **Remaster Instances from FRAM to CEL** to change (remaster) the instance reference from IC Compiler FRAM view to IC Compiler CEL view during the import operation.
4. In the **Options** tab (shown in [Figure 14](#)), specify optional details.

## Chapter 2: Import and Export

### Importing Design Data from IC Compiler

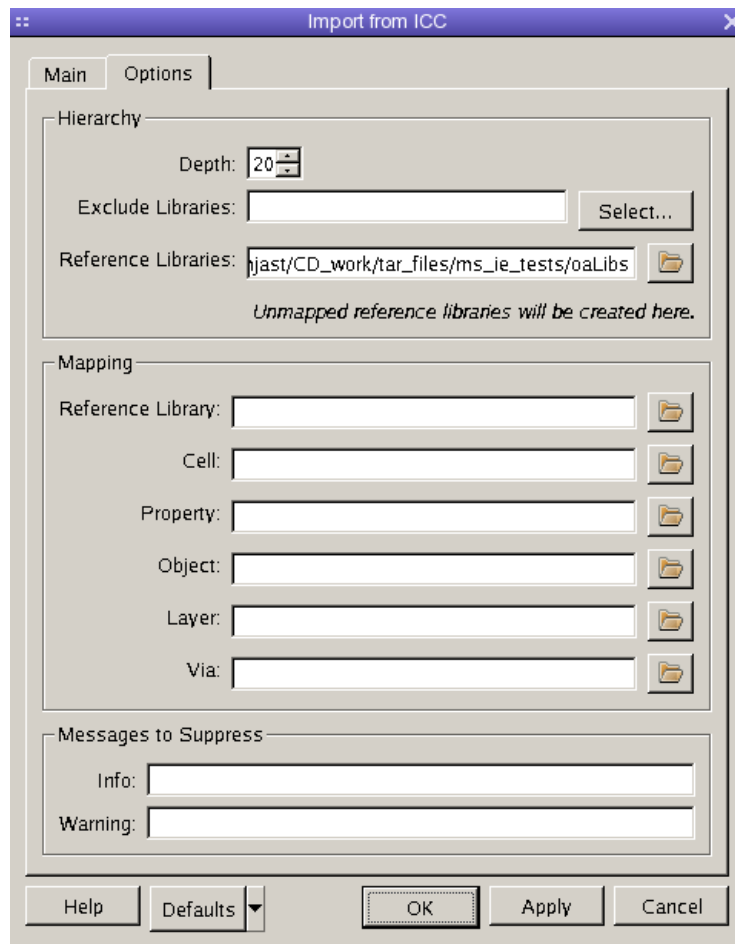


Figure 14 Import from ICC Dialog: Options Tab

- (Optional) In the **Hierarchy** region of the dialog box, use the **Depth** option to limit how much of the design information is imported.  
For any number greater than 0, the **Exclude Libraries** option is enabled.
- (Optional) Specify any libraries you want to exclude from the import operation in the **Exclude Libraries** option.
- (Optional) Use the **Reference Libraries** option to specify the path to a target directory to create any unmapped IC Compiler libraries referenced in the design that you want to import.  
Accept the default directory path or enter a new path.



- d. In the **Mapping** region of the dialog box, specify optional mapping details.

(Optional) As you prefer, enter full paths for the **Reference Library**, **Cell**, **Property**, **Object**, **Layer**, and **Via** you want to map.

Adding this information makes it possible to include instances in your design project that reference masters in external libraries. To instantiate an instance that references an external library, you can map an external library to the library that will be used in the target project.

The system looks for your user-provided mapping information when you start an import or export operation. If your user-provided information is found, the system writes that information to the following mapping files in the `iccbridge` directory: [layer.map](#), [via.map](#), [lib.map](#), [cell.map](#), [property.map](#), [object.map](#).

- e. (Optional) In the **Messages to Suppress** region of the dialog box, enter space-separated lists of the **Info** and **Warning** message IDs that you want to suppress.

5. Click **OK**.

The import process creates the `iccbridge` run directory at the path `$SYNOPSYS_CUSTOM_LOCAL/iccbridge` and saves the current state to `prefs.xml` within the `iccbridge` directory.

If the source library is new, the import process also optionally creates a technology file and a display resources file. It creates the Custom Designer library and imports technology from either the input IC Compiler library or an existing user-specified Custom Designer platform technology file.

The process completes by optionally loading display resources from the IC Compiler design.

6. Add access to this imported library by updating the library definition file through the Library Manager.

See also

For details about object mappings, see [IC Compiler - Custom Designer Platform Library Mapping Files on page 30](#)

For instructions for creating target libraries, see [Creating Target Libraries for IC Compiler Import on page 31](#)

## IC Compiler - Custom Designer Platform Library Mapping Files

For reference while importing or exporting libraries, use this section to view mapping files for IC Compiler libraries and Custom Designer platform libraries.

See the tables that follow for details about the library import and export files.

[Table 2 on page 30](#) lists cell map details.

Table 2 *cell.map*

Custom Designer	Custom Designer	Custom Designer	IC Compiler	IC Compiler	IC Compiler
library name	cell name	view name	library name	cell name	view name
string	string	string	string	string	string

[Table 3 on page 30](#) lists layer map details.

Table 3 *layer.map*

Custom Designer	Custom Designer	IC Compiler	IC Compiler	Custom Designer	Custom Designer	IC Compiler
layer name <sup>1</sup>	purpose name <sup>1</sup>	layer number <sup>1</sup>	dataType number <sup>1</sup>	material name	mask number	mask name
string	string	int	int	enum <sup>2</sup>	int	string

1. Required

2. The possible values are: *nWell, pWell, nDiff, pDiff, nImplant, plmplant, poly, cut, metal, contactlessMetal, diffusion, recognition, other*. Use the option *other* when the material for the layer is not specified.

[Table 4 on page 30](#) lists library map details.

Table 4 *lib.map*

Custom Designer	IC Compiler
library name	<path>/library name
string	string

[Table 5 on page 31](#) lists object map details for these objects: oaAreaBlockage, oaAreaHalo, oaLayerBlockage, oaLayerHalo, oaAreaBoundary, oaClusterBoundary, oaPRBoundary, oaSnapBoundary.

Table 5 *object.map*

Custom Designer	Custom Designer	IC Compiler	IC Compiler
object type	layer	layer number	layer name
enum	string	int	string

[Table 6 on page 31](#) lists property mapping details.

Table 6 *property.map*

Custom Designer	IC Compiler
property name	property name
string	string

[Table 7 on page 31](#) lists via mapping details.

Table 7 *via.map*

Custom Designer	IC Compiler
viaDef name	contactCode name
string	string

See also

For instructions for creating target libraries, see [Creating Target Libraries for IC Compiler Import on page 31](#)

More about [Import and Export on page 25](#)

For a description of the mapping files that can be provided when using an attached OA Tech library, see [Using Separate OpenAccess Technology for an Opened IC Compiler Library on page 160](#).

## Creating Target Libraries for IC Compiler Import

If you have not created a target library to receive imported data from IC Compiler, use the following procedure to create the library.

## Chapter 2: Import and Export

### Importing Design Data from IC Compiler

When the IC Compiler tool to Custom Designer platform import operation completes, the system writes the converted data to a Custom Designer library (the target library), overwriting any existing information.

To create a target library for IC Compiler import:

1. Choose **File > Import > From ICC** from the console to open the **Import from ICC** dialog box.
2. In the **Import from ICC** dialog box, click **Create**.

The **Create Library for Import** dialog box opens (Figure 15).

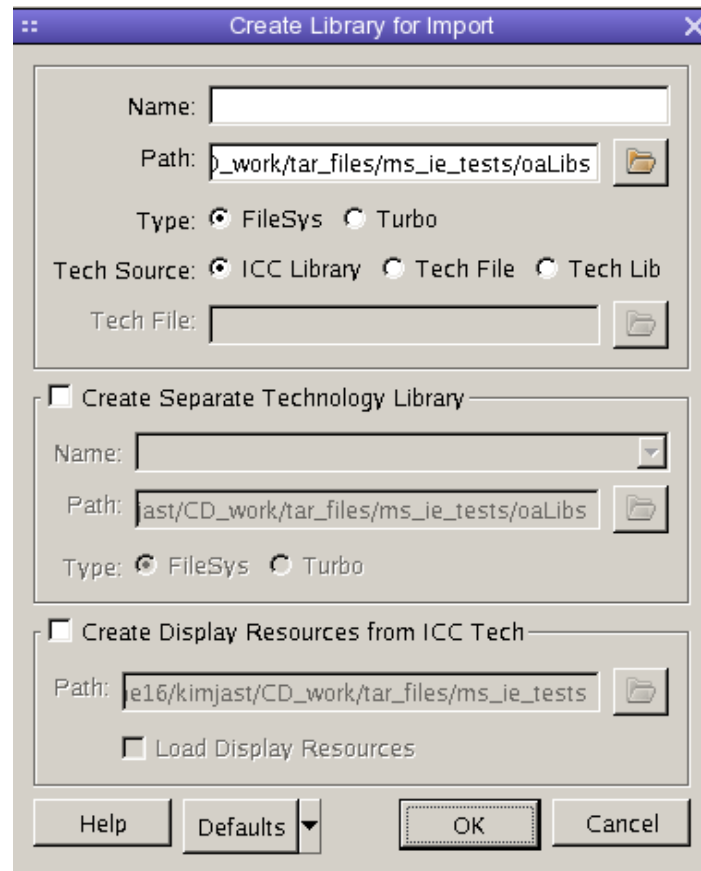


Figure 15 Create Library for Import Dialog

3. Specify a library **Name**.

The system creates this library during the import operation.

4. Specify a **Path** to the library.
5. Accept the default **Type** (FileSys).

6. Select a **Tech Source** for the technology database for the new library.
  - a. Selecting the **ICC Library** (default) option creates a Custom Designer technology database based on the technology information contained in the IC Compiler library.
  - b. If you select the **Tech File** option, also specify the path to the technology file.
  - c. If you select the **Tech Lib** option, also choose an existing library from the **Tech Library** menu.
7. (Optional) If you want to create a second library to contain the imported technology, enable the **Create Separate Technology Library**, and specify the required information.
8. (Optional) If you want the system to create a Display Resources file (.drf or .tcl file), enable the option to **Create Display Resources from ICC Tech**, and specify the path name.
9. (Optional) If you want the Custom Designer platform to load the display resources file after the import operation, enable the **Load Display Resources** option.
10. Click **OK**.

The system creates the library, the dialog box closes, and the new library name now appears in the Output region of the **Import from ICC** dialog box.

After you create the target library, you can continue the process of [Importing Design Data from IC Compiler on page 25](#).

See also

For details about object mappings, see [IC Compiler - Custom Designer Platform Library Mapping Files on page 30](#)

For instructions for creating target libraries, see [Creating Target Libraries for IC Compiler Import on page 31](#)

More about [Import and Export on page 25](#)

---

## Importing Individual Designs from IC Compiler

You can import individual designs from IC Compiler after you set up the import process by performing it once. Follow the procedure for [Importing Design Data from IC Compiler on page 25](#).

After you import a design once, you can re-import it to discard edits and revert the design to its last imported state.

To import an individual design:

1. In any design window, choose **Tools > ICC Link** to add the ICC menu choice to the design menu.
2. Choose **ICC > Import** to open the **Import from ICC** dialog.

See also

More about [Import and Export on page 25](#)

---

## Exporting Designs and Libraries to IC Compiler

When you are co-designing with the IC Compiler tool and the Custom Designer platform, you can export designs for input to the IC Compiler tool in several ways.

You can export designs and their libraries, or save and export a design only. For either of these options, use one of the procedures in this section.

When you have added timing shell black boxes to a design, you can export it as a VerilogICC netlist, either with single or multiple power supplies. For this option, see [Exporting Design Data in Netlist Format](#) and choose the VerilogICC netlist option.

These sections cover:

- [Exporting Libraries to IC Compiler on page 34](#)
- [Exporting Individual Designs to IC Compiler on page 43](#)

---

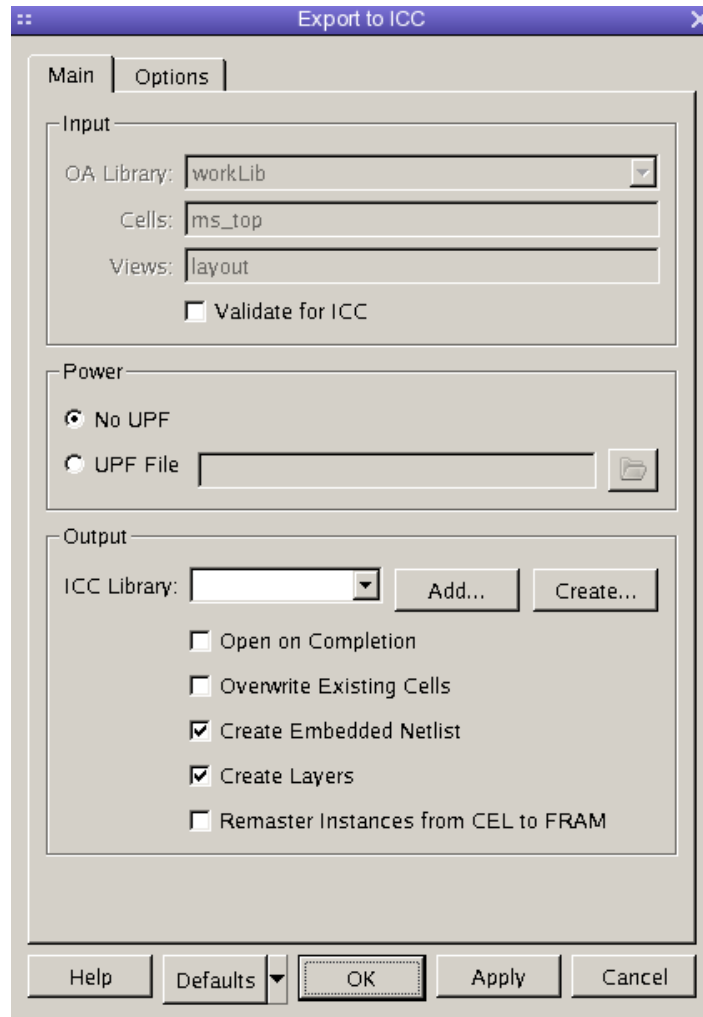
## Exporting Libraries to IC Compiler

Export libraries to the IC Compiler tool for co-design using the Custom Designer platform and the IC Compiler tool. When you want to use the IC Compiler tool, you can export the design and its libraries.

When you export designs to IC Compiler, a bridge process between the two tools takes the source library containing the data you want to export, converts it for IC Compiler tool access, and adds it to a target library for your use with the IC Compiler tool.

To export a library to IC Compiler:

1. From the Console, choose **File > Export > To ICC**, or use the Tcl command **xt::exportToICC**. The **Export to ICC** dialog opens ([Figure 16](#)).



*Figure 16 Export to ICC Dialog: Main Tab*

## Chapter 2: Import and Export

### Exporting Designs and Libraries to IC Compiler

2. On the **Main** tab, specify source and destination details as described in [Table 8 on page 36](#).

Table 8 *Export to ICC (Main)*

Setting	Description
<b>Input</b>	<p><b>OA Library</b> is the source library you want to translate.</p> <p><b>Cell(s)</b> is the space-separated list of cells you want to translate. If you leave this field blank, the system translates all cells in the library.</p> <p><b>View(s)</b> is the space-separated list of view types you want to translate. If you leave this field blank, the system translates all views.</p> <p>Choose <b>Validate for ICC</b> to run the validation process for each design. If any of them fail, export is cancelled. If there are only warnings in the design, the export process is not interrupted.</p>
<b>Power</b>	Choose <b>No UPF</b> or browse to your <b>UPF File</b> .
<b>Output</b>	<p><b>ICC Library</b> is the destination IC Compiler library. To populate this field, click <b>Add</b> to open the <b>Add Existing ICC Library</b> dialog or <b>Create</b> to open the <b>Create ICC Library for Export</b> dialog (see <a href="#">Creating IC Compiler Libraries for Export on page 40</a>). Or click the drop-down menu to enable the <b>Other Library</b> option and then browse to select it.</p> <p><b>Open on Completion</b> opens the target IC Compiler design in Custom Designer upon successful completion of the export and any optional post-processing script.</p> <p><b>Overwrite Existing Cells</b> translates existing cells in the IC Compiler design. If you do not choose this, existing cells are not translated.</p>



*Table 8    Export to ICC (Main)*

Setting	Description
	<p><b>Create Embedded Netlist</b> IC Compiler maintains both a flat and a hierarchical representation of the design for the implementation flows. The hierarchical representation is the logical netlist, which is stored as an internal data structure called hierarchy preservation.</p> <p>When this option is not selected, the physical representation of the design along with connectivity is exported to IC Compiler. You can create a FRAM view for this block in IC Compiler, but you cannot run a Place and Route flow. Choosing this option creates the logical netlist representation in the target IC Compiler design, which enables implementation flows on the IC Compiler design without requiring a separate Verilog Import process in IC Compiler. Contact Synopsys support for more information</p> <p><b>Create Layers</b> Exports shapes on layers not referenced by the technology. If this option is left unchecked, those shapes will be skipped.</p> <p><b>Remaster Instances from CEL to FRAM</b> By default the instances of the target design are bound to the same view as the source design.</p> <p>Choosing this option forces all instances that are bound to a layout view to use the FRAM view as reference instead of CEL view. The FRAM view should exist for the instance's master. If the FRAM view does not exist, the instance will be bound to a CEL view.</p> <p>If you want to change the reference views for few instances, use the cell mapping file option.</p>

3. Change to the **Options** tab in the **Export to ICC** dialog ([Figure 17](#)).

## Chapter 2: Import and Export

### Exporting Designs and Libraries to IC Compiler

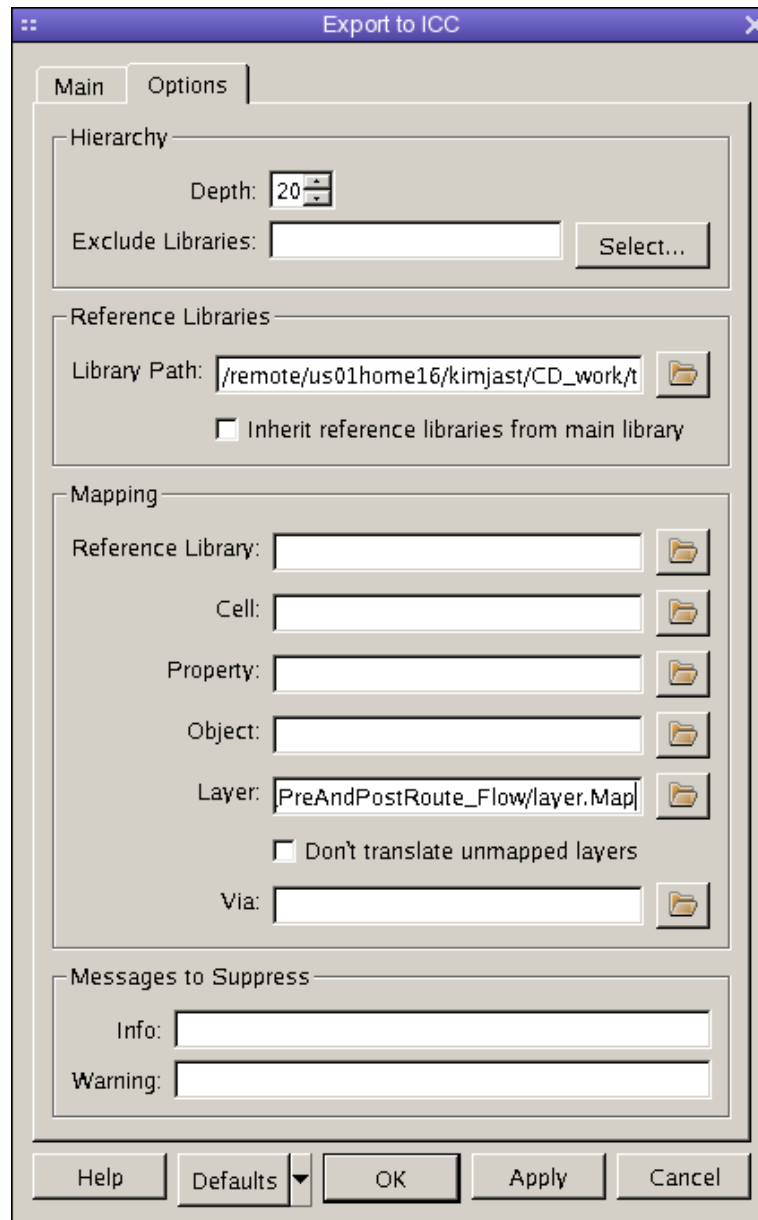


Figure 17 Export to ICC Dialog: Options Tab

On the **Options** tab, specify option details as described in [Table 9 on page 39](#).

*Table 9    Export to ICC (Options)*

Setting	Description
<b>Hierarchy</b>	<p><b>Depth</b> limits the amount of design information exported.</p> <p><b>Exclude Libraries</b> if depth is greater than 0, you can choose to exclude libraries you enter here from the export.</p>
<b>Reference Libraries</b>	<p><b>Library Path</b> browse to the directory where you want unmapped reference libraries to be created. You can choose whether or not to <b>Inherit reference libraries from main library</b>.</p>
<b>Mapping</b>	<p><b>Reference Library</b> maps using the external reference <code>lib.map</code> file you browse to select.</p> <p><b>Cell</b> maps using the <code>cell.map</code> file you browse to select.</p> <p><b>Property</b> maps using the <code>property.map</code> file you browse to select.</p> <p><b>Object</b> maps using the <code>object.map</code> file you browse to select.</p> <p><b>Layer</b> maps using the <code>layer.map</code> file you browse to select. Choose whether or not to select <b>Don't translate unmapped layers</b>.</p> <p><b>Via</b> maps using the <code>via.map</code> file you browse to select.</p>
<b>Messages to Suppress</b>	<p><b>Info</b> Is a space-separated list of Information message IDs you want the tool to suppress</p> <p><b>Warning</b> Is a space-separated list of Warning message IDs you want the tool to suppress</p>

- The **Mapping** section of the **Options** tab explicitly specifies the library name mappings using full path names.

## Chapter 2: Import and Export

### Exporting Designs and Libraries to IC Compiler

To view details about all mappings between IC Compiler and Custom Designer platform libraries, see [IC Compiler - Custom Designer Platform Library Mapping Files on page 30](#).

The tool reads the run directory map files, and uses them to perform translation. Duplicate mappings are ignored. Any mapping that conflicts with a previously read mapping generates a warning message describing the change the tool detects. When the tool detects a conflict, it uses the latest mapping found in the translation.

Upon completion of the translation, the tool writes the full mapping back to the run directory, overwriting any currently saved files.

5. Click **OK**.

The system sets up an `iccbridge` run directory at the path `$SYNOPSYS_CUSTOM_LOCAL/iccbridge`.

It saves the current preference settings to `prefs.xml` within the `iccbridge` directory.

If the target IC Compiler library is new, the system optionally creates a technology file, and creates a target IC Compiler library when it launches the export operation.

The Custom Designer platform creates hierarchy preservation within the exported design based on the source design.

See also

More options for design and data [Import and Export on page 25](#)

---

## Creating IC Compiler Libraries for Export

If you have not created an output IC Compiler library, do so from the **Export to ICC** dialog. To create a target library for export to IC Compiler:

1. Choose **ICC > Export** to open the **Export to ICC** dialog box.
2. In the **Export to ICC** dialog box, click **Create**.

The **Create ICC Library for Export** dialog box opens (Figure 18).

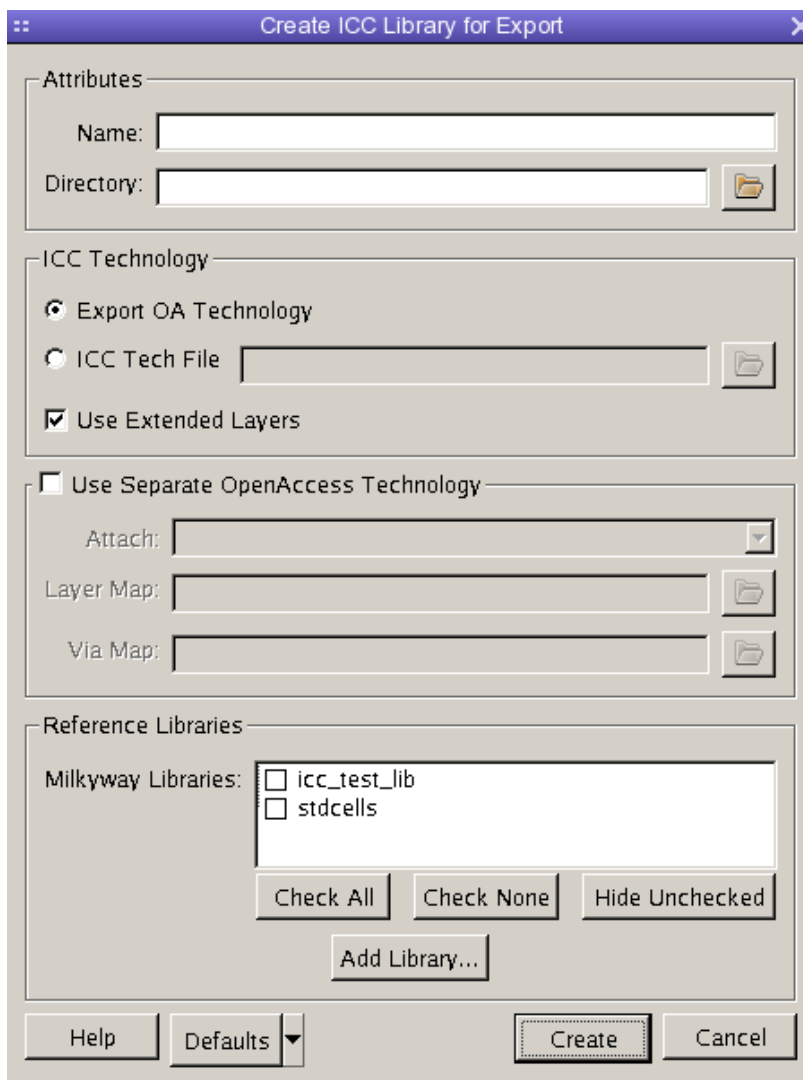


Figure 18 Create ICC Library for Export Dialog

3. Specify a library **Name**.  
The system creates this library during the export operation.
4. Specify or browse to a **Directory**.
5. In the ICC Technology section, **Export OA Technology** is on by default. **ICC Tech File** uses the technology file you browse to open. Check **Use Extended Layers** to enable use of extended layers.

## Chapter 2: Import and Export

### Exporting Designs and Libraries to IC Compiler

6. (Optional) **Use Separate OpenAccess Technology** then choose a library to attach. **Attach** uses OpenAccess technology from the library you choose by attaching it, and does not enable two-way technology synchronization between IC Compiler and OpenAccess technology representations.

These mapping files are used to map the layers and vias from the OA technology library to the corresponding layers and vias defined in the IC Compiler technology. This is needed if you supplied an IC Compiler technology file and that technology file has different layer names and numbers compared to the OA technology library that you want to use, too.

7. (Optional) The **Reference Libraries** section lists all open ICC libraries. Open additional libraries by clicking **Add Library**, which opens the **Add Existing ICC Library** dialog (see [Designs Originating with the IC Compiler Tool on page 51](#)). None of the libraries are checked by default.

The new library is not actually created upon clicking **Create**. Instead, the information is captured and the new library is listed in the **Milkyway Libraries** box in the **Create ICC Library for Export** dialog with the suffix "(new)" (such as "MyLibName (new)") to differentiate it from existing libraries.

To open an existing ICC library, click **Add**, which brings up the **Add Existing ICC Library** dialog (see [Designs Originating with the IC Compiler Tool on page 51](#)). Upon dismissing this dialog, the list of ICC libraries is refreshed.

8. Click **OK**.

The system creates the library, the dialog box closes, and the new library name now appears in the **Output** region of the **Export to ICC** dialog box.

After you create the target library, you can continue the process of [Importing Design Data from IC Compiler on page 25](#).

See also

For details about object mappings, see [IC Compiler - Custom Designer Platform Library Mapping Files on page 30](#)

For a description of the mapping files that can be provided when using an attached OA Tech library, see [Using Separate OpenAccess Technology for an Opened IC Compiler Library on page 160](#).

More about [Import and Export on page 25](#)

## Exporting Individual Designs to IC Compiler

You can export individual designs to IC Compiler after you set up the export process by performing it once. Follow the procedure for [Exporting Libraries to IC Compiler on page 34](#).

This provides flexibility to export the current design to IC Compiler.

To export an individual design:

1. In any design window, choose **Tools > ICC Link** to add the ICC menu choice to the design menu.
2. Choose **ICC > Export** to open the **Export to ICC** dialog.

Alternatively, to save and export an individual design:

- Choose **ICC > Save and Export** to open the **Export to ICC** dialog.

See also

More about [Import and Export on page 25](#)

## **Chapter 2: Import and Export**

Exporting Designs and Libraries to IC Compiler



*Information about opening designs and deciding which editor to use.*

For extended information on opening designs and general designing, see the *Platform User Guide*.

These sections cover:

- [Opening a Design From the Library Manager on page 45](#)
- [View Mapping between IC Compiler and Custom Designer on page 46](#)
- [Working on Schematic Designs on page 47](#)
- [Working on Layout Designs on page 47](#)
- [Closing Designs on page 47](#)

---

## Opening a Design From the Library Manager

Open a design cellView to view or edit it. To open a design from the Library Manager:

1. In the Library Manager, specify the names of the library, category (optional), cell and view.  
  
Entering text in an input field at the top of a column filters the set of names offered. To return to the full list, delete text in the input field.
2. On the view name, either double-click, or use the alternate mouse button and choose **Open**.

The design opens in the editor for this view.

Preference options apply. Active cellView level preference settings also apply, and take precedence.

## Chapter 3: Designing

### View Mapping between IC Compiler and Custom Designer

The first time an IC Compiler design is opened in Custom Designer, the design information is imported to the custom environment. The duration of this step depends on the size of the design, the number of shapes in the design, and how many cell masters are instantiated. If the IC Compiler design is not modified, subsequent design opens are faster.

Opening an IC Compiler design in Custom Designer places a lock on the design so that it cannot be modified in both environments at the same time.

---

## View Mapping between IC Compiler and Custom Designer

The following table shows layout view mapping between IC Compiler and Custom Designer.

*Table 1 View Mapping between IC Compiler and Custom Designer*

IC Compiler Design viewName	Custom Designer View Type
*layout*	.CEL
*abstract*	.FRAM
*fill*	.FILL
*smash*	.SMASH
*ILM*	.ILM

The view you open determines the editor you will use. All of the above views use the Custom Designer Layout Editor. In Chapter 5 of the *Galaxy Custom Designer Platform User Guide*, the section "Deciding Which Editor to Use" lists design view types, the data they act on, and where to find user assistance for editing.

Opening an IC Compiler design in Custom Designer that has the instances bound to a FRAM view in IC Compiler automatically binds the instances to an abstract view (the equivalent of the FRAM view).

To swap the views for any instance, run the `change_macro_view` command in IC Compiler to change the reference for the desired instances. Save the design in IC Compiler and then reopen the design in Custom Designer.

## Working on Schematic Designs

For detailed information on working with schematic designs, refer to the *Schematic Editor User Guide*.

---

## Working on Layout Designs

For detailed information on working with layout designs, refer to the *Layout Editor User Guide*.

---

## Closing Designs

Once your editing steps are completed in Custom Designer, simply save the design to save these edits back to the IC Compiler view.

After you draw routes, you can use the IC Compiler tool to perform place and route on the complete design. You can then either close the design in Custom Designer or make it read-only to use the IC Compiler tool to perform further design implementation tasks such as floor plan, placement, routing, and design optimizations.

You can go back and forth between the two design environments as many times as needed to complete both the custom/analog layout and digital layout implementation.



## Preparing Libraries for Co-Design

---

*How to prepare libraries for co-design on the Custom Designer platform.*

You must prepare libraries for co-design with the IC Compiler tool on the Custom Designer platform before designing analog blocks or implementing digital blocks.

- If no IC Compiler library exists for your design, create a library of the IC Compiler type on the Custom Designer platform.  
For instructions, see [Designs Originating on the Custom Designer Platform on page 49](#).
- If a IC Compiler library does exist for your design, make it available to the Custom Designer platform by adding its library.  
For instructions, see [Designs Originating with the IC Compiler Tool on page 51](#).
- If no timing library exists for your custom design, and you are planning to run place and route with multi-voltage power sources, you need to create one.  
For instructions, see [Creating Timing Libraries for Multi-Voltage Power in Place and Route on page 53](#).

---

## Designs Originating on the Custom Designer Platform

When you have a design originating on the Custom Designer platform, and you want to transfer it to the IC Compiler tool for digital block place and route, create an IC Compiler library for it.

(If this library already exists, see instead [Designs Originating with the IC Compiler Tool on page 51](#).)

## Chapter 4: Preparing Libraries for Co-Design

Designs Originating on the Custom Designer Platform

To create an IC Compiler design library:

1. From the Custom Designer platform, open the Library Manager tool.
2. In the Library Manager, create a new library.
3. Choose the library type **ICC**.
4. Define the library as described in [Table 1 on page 50](#).

*Table 1 IC Compiler Library Settings*

Settings	Description
<b>Ref Libs</b>	Shows the list of open IC Compiler reference libraries that this library will have as reference libraries. Use the check boxes to make selections. You can choose from this list, or you can open more libraries to consider for use as reference libraries.
<b>ICC Technology</b>	<b>ICC Tech File</b> – Uses the technology file you browse to open. <b>Bus Naming Style</b> – Choose the bus naming style.
<b>Use Separate OA Technology</b>	<b>Attach</b> – Uses OpenAccess technology from the library you choose by attaching it, and does not enable two-way technology synchronization between IC Compiler and OpenAccess technology representations.  Optional, in case an IC Compiler technology file is not available. Browse to your <b>Layer Map</b> and <b>Via Map</b> files to map the OA layers from the OA technology data to different Milkyway layers in the Milkyway database.

5. Click **OK**.

See also

More about [Preparing Libraries for Co-Design](#) on page 49.

---

## Designs Originating with the IC Compiler Tool

When you have a design originating with the IC Compiler tool, and you want to develop it on the Custom Designer platform, add an IC Compiler design library for it.

To add an IC Compiler design library:

1. In the Custom Designer Library Manager, either choose **File > Add ICC Library** or right-click anywhere in the **Libraries** list, and choose **Add**.
2. In the **Add Existing ICC Library** dialog box (shown in [Figure 1](#)), ignore the **Name** field, and browse to choose the IC Compiler design library for this design.

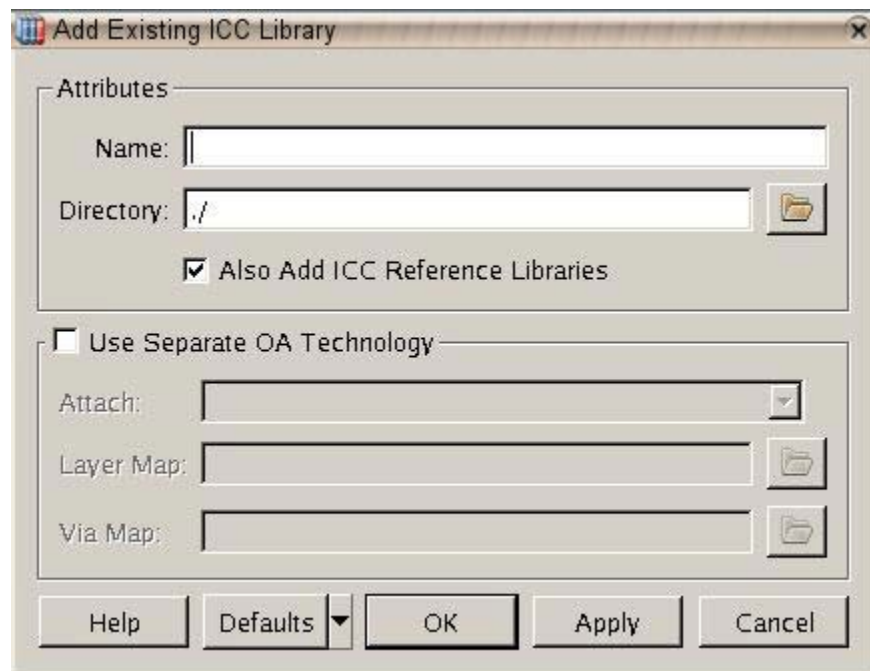


Figure 1 Add Existing ICC Library Dialog

The *Add Existing ICC Library* dialog includes the following options and fields:

- **Name:** Specify the name of the target IC Compiler library in this text field. This field automatically populates when you browse to the IC Compiler library directory in the **Directory** field.
- **Directory:** Specify the path to the target IC Compiler library's directory with its full path in this text field, or use the **Browse** button to select the desired directory.
- **Also Add ICC Reference Libraries:** On by default. When selected, Custom Designer adds all reference libraries stored within the specified IC Compiler library to the library definitions and updates libraries referenced by this design when the design changes.

When the tool encounters duplicate names for reference libraries, it appends **\_ICC** to the duplicate, and adds a number to increment each duplicate name.

- **Use Separate OA Technology:** When selected, you can use the **Attach** field to select a separate OA technology file from your library. This OA technology file will be used with OA applications but not IC Compiler. You must also specify the layer and via mapping files using **Layer Map** and **Via Map** to import the IC Compiler layers and vias to the corresponding OA layers and vias defined in the associated OA technology library.

**Note:** Using separate OA technology requires write access to the IC Compiler library.

3. (Optional) Choose to **Also Add ICC Reference Libraries** to update libraries referenced by this design when the design changes.
4. Click **OK**.

The tool adds a DEFINE statement with the path to this library to the current `lib.defs` file for automatic access on next startup.



See also

More about [Preparing Libraries for Co-Design on page 49](#).

---

## Creating Timing Libraries for Multi-Voltage Power in Place and Route

You can create timing libraries when you want to run place and route on designs with multiple power supplies using power supply definitions or a UPF file.

To create a timing library:

1. On the Custom Designer platform, export the design in Liberty library file format.  
  
For instructions, see the section "Exporting Design Data in Liberty Library File Format" in Chapter 4 of the *Galaxy Custom Designer Platform User Guide*.
2. Convert the Liberty format file from `<cellName>.lib` to `<cellName>.db` using one of these two methods:
  - Use the IC Compiler tool to make this conversion, using the IC Compiler functions `read_lib` and `write_lib`.
  - Set the `leICCGenerateBlackBoxTimingFiles` preference to *true* for the place and route process.

When this preference is set to *true*, the tool searches for a `<blackbox>.lib` file in the UNIX paths specified in the search path preference `leICCSearchPath`. If it finds the a `.lib` file, it converts this file to a `<blackBox>.db` file.

3. Supply this file as input to the place and route process.

See [Running IC Compiler Place and Route from Design Schematics or Verilog Netlists on page 91](#).

See also

More about [Preparing Libraries for Co-Design on page 49](#).

## **Chapter 4: Preparing Libraries for Co-Design**

Creating Timing Libraries for Multi-Voltage Power in Place and Route

## Editing an IC Compiler Design in a Custom Designer Environment

---

*How to edit an IC Compiler design in Custom Designer.*

Analog content in IC Compiler designs is increasing and designers have a need to use a full custom layout editor and custom router to implement efficiently the analog nets in those designs. You can develop analog nets with co-design using the Custom Designer platform and the IC Compiler tool.

You can implement analog routes, either automatically or interactively,

After you draw routes, you can use the IC Compiler tool to perform place and route on the complete design.

These sections cover:

- [Custom Designer Tools Used in IC Compiler Co-design Flows](#)
- [Implementing Analog Nets](#)
- [Performing IC Compiler Operations on Design Layouts](#)
- [Use Model Examples](#)

---

### Custom Designer Tools Used in IC Compiler Co-design Flows

The following sections briefly describe some Custom Designer tools you might use in an ICC-CD flow.

## Design Navigator

You can use the Design Navigator to view layout information such as nets, instances, instTerms, Terms, net groups, and figure groups from the IC Compiler design data. Constraints can be created using the Constraint Manager on nets selected in the Design Navigator.

Access the Design Navigator using **Window > Assistants > Design Navigator** from the layout window.

For more details on the Design Navigator, see Chapter 12 of the *Galaxy Custom Designer Platform User Guide*.

---

## Constraint Editor

You can manually set constraints by using the Constraint Editor assistant, which can be accessed using **Window > Assistants > Constraint Editor** in the *Layout* or *Schematic Editor* windows. For more details on working with design constraints, see Chapter 7 of the *Galaxy Custom Designer Layout Editor User Guide*.

---

## Check EM and Check Resistance

You can use the Design Navigator to check electromigration and resistance for selected nets or all the nets in a design. For more details, see the section "Checking Electromigration and Resistance" in the *Galaxy Custom Designer Layout Editor User Guide*.

---

## Create Interconnect

IC Compiler designs have wire tracks created automatically as part of the floorplan setup. The wire tracks are used by the IC Compiler auto router ZRoute to route the design efficiently. When manually editing IC Compiler designs, it is preferable to keep the routes on track to keep the efficiency of the layout, leave tracks available for other routes as far as possible, and also as an aid to keep the layout DRC clean.

Custom Designer supports displaying the wire tracks using the Track Pattern assistant, which has an option to support snapping routes to tracks during the **Create > Interconnect** command. Open the Track Pattern assistant using

**Window > Assistants > Track Pattern Assistant.** In the Track Pattern assistant, enabling the Active setting for a layer sets the track pattern to be active. For an active track pattern, mouse movements snap to the track during creation. Copy, move, and stretch operations also snap to the track. (For more details on the Track Pattern assistant, see Chapter 12 of the *Galaxy Custom Designer Platform User Guide*.)

IC Compiler prefers to deal with HWIRE and VWIRE objects, for instance for eco routing. For more flexibility in the interaction between IC Compiler and Custom Designer, you can set the default objects that will be created by the custom router, Interactive Router, and **Create > Interconnect** commands to be `oaPathSeg`. To do so, set the following preference:

```
db::setPrefValue leUsePathSeg -value true
```

The preference value can be changed in the **Options > Design** dialog, Command tab.

---

## Implementing Analog Nets

Before you perform place and route for an entire design with the IC Compiler tool, you can choose to draw individual routes for analog nets on the Custom Designer platform. This includes nets with special shielding or contours.

You can automatically or interactively draw routes for individual nets.

Route guides you have defined using the IC Compiler tool for regions where drawing routes will not be permitted will show as area boundaries in the Custom Designer Layout Editor tool.

Foundry rules are applied in an equivalent manner in both tools.

These sections cover:

- [Safe Editing](#)
- [Namespace Mapping](#)
- [Object Mapping](#)
- [Adjusting Placement of Cells](#)
- [Drawing Routes Automatically](#)
- [Drawing Routes Interactively](#)

---

## Safe Editing

The Custom Designer safe editing mode prevents you from modifying the logical elements of the design that would affect the IC Compiler view consistency check. It ensures that you can safely continue to use IC Compiler for the design implementation after editing it in Custom Designer.

When working with IC Compiler designs, Custom Designer's safe editing mode is on by default.

---

## Namespace Mapping

When IC Compiler designs open on the Custom Designer platform, IC Compiler tool instance and net names map to Custom Designer platform naming conventions, which use OpenAccess notation.

The following table shows the character mapping between IC Compiler and Custom Designer. Unless otherwise indicated, the information below is true for both instance and net names.

IC Compiler	Custom Designer
	/
#2f and /	<sup>1</sup> (see Note)
[	( for instances < for nets
]	) for instances > for nets
<	#3c
>	#3e
\	#5c

*1. Both #2f and / map from IC Compiler to Custom Designer as |, but from Custom Designer to IC Compiler, | only maps to /.*

## Object Mapping

The following table shows object mapping between IC Compiler and Custom Designer.

**Note:** Objects labelled cd\* are custom objects and are warehoused in IC Compiler for roundtrip.

IC Compiler	Custom Designer
Cell	oaInst
Pin	oaPin
Port	oaBitTerm
Net	oaNet
Metal Layer Blockages	oaLayerBlockage of type routing
System Metal Layer Blockages <sup>1</sup>	oaLayerBlockage of type routing
Placement Blockages	oaAreaBlockage
Rectangle	oaRect
Polygon	oaPolygon
Trapezoid	oaPolygon
Text	oaText
Path <sup>2</sup>	oaPath
HWIRE/VWIRE	oaPathSeg <sup>3</sup>
Parameterized contact codes	oaStdVia
Rectangle-based contact codes	oaStdVia/oaCustomVia
Design-specific via master	oaStdVia/oaCustomVia

## Chapter 5: Editing an IC Compiler Design in a Custom Designer Environment

### Implementing Analog Nets

IC Compiler	Custom Designer
Voltage Area	leVoltageArea <sup>3</sup>
Routing Corridors	leRouteCorridor <sup>4</sup>
Wire Tracks	leTrackPatternDef/leTrackPattern
cdTrackPatternDef	leTrackPatternDef/leTrackPattern originating in Custom Designer
Core Area	oaGroup <sup>5</sup>
Cell Row (Site Row)	oaRow
Cell Boundary	oaPRBoundary
PR Boundary	oaSnapBoundary
Route Guides	oaAreaBoundary
cdAreaBoundary	oaAreaBoundary originating in Custom Designer
Custom Designer-Specific Objects	
cdSteiner	Steiners
Group	oaRoutes
Edit Group	Striped Wires
Edit Group	Synchronous Fig Groups



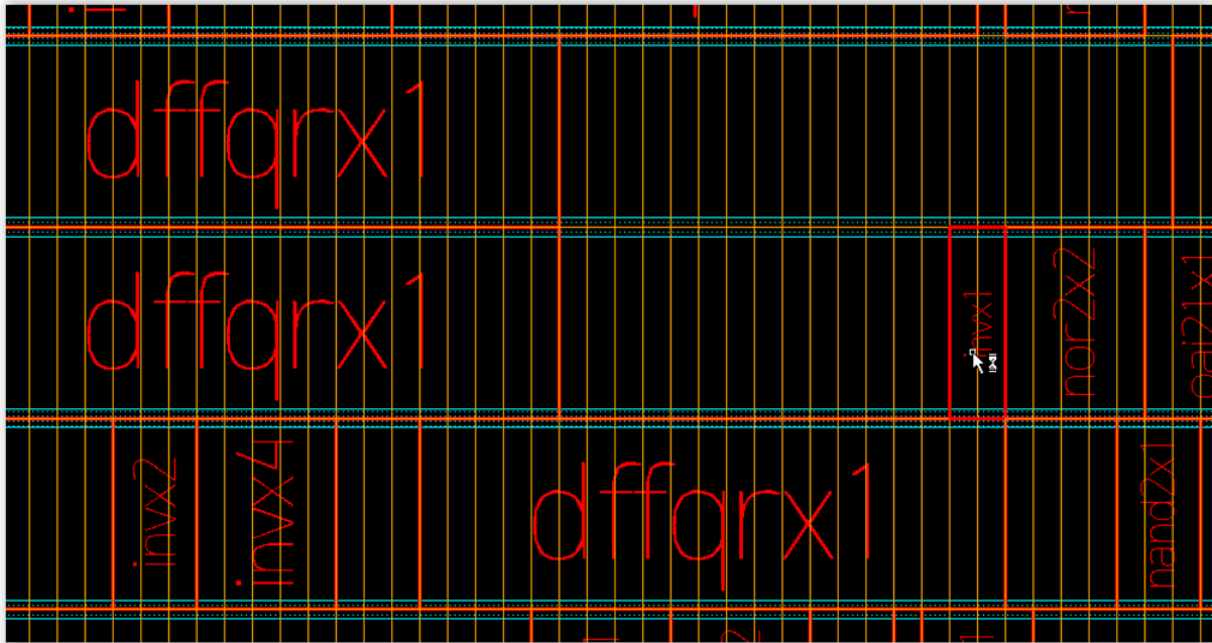
IC Compiler	Custom Designer
Edit Group	Multi-Part Paths
Polygon	oaEllipse <sup>6</sup>

1. IC Compiler via layer blockages in FRAM view are mapped to via layers in Custom Designer.
2. Shield association to nets is not maintained in IC Compiler when path objects are used.
3. IC Compiler prefers to deal with HWIRE and VWIRE objects, for instance for eco routing. For more flexibility in the interaction between IC Compiler and Custom Designer, you can set the default objects that will be created by the custom router, Interactive Router, and Create > Interconnect commands to be oaPathSeg.  
To do so, set the following preference:  
`db::setPrefValue leUsePathSeg -value true`  
The preference value can be changed in the Options > Design dialog, Command tab.
4. A new semantic object in Custom Designer.
5. The oaGroup mapped to the core area uses the oaFlatGroupDef named 'ICC\_BASE\_ARRAY', which identifies the group semantically and only allows oaRow objects to be members of the group.
6. IC Compiler does not support polygons, so oaEllipse(s) are translated to 32-point IC Compiler polygons. Ellipses are recovered in Custom Designer when opening the IC Compiler design even if they are stored as polygons in the Milkyway database.

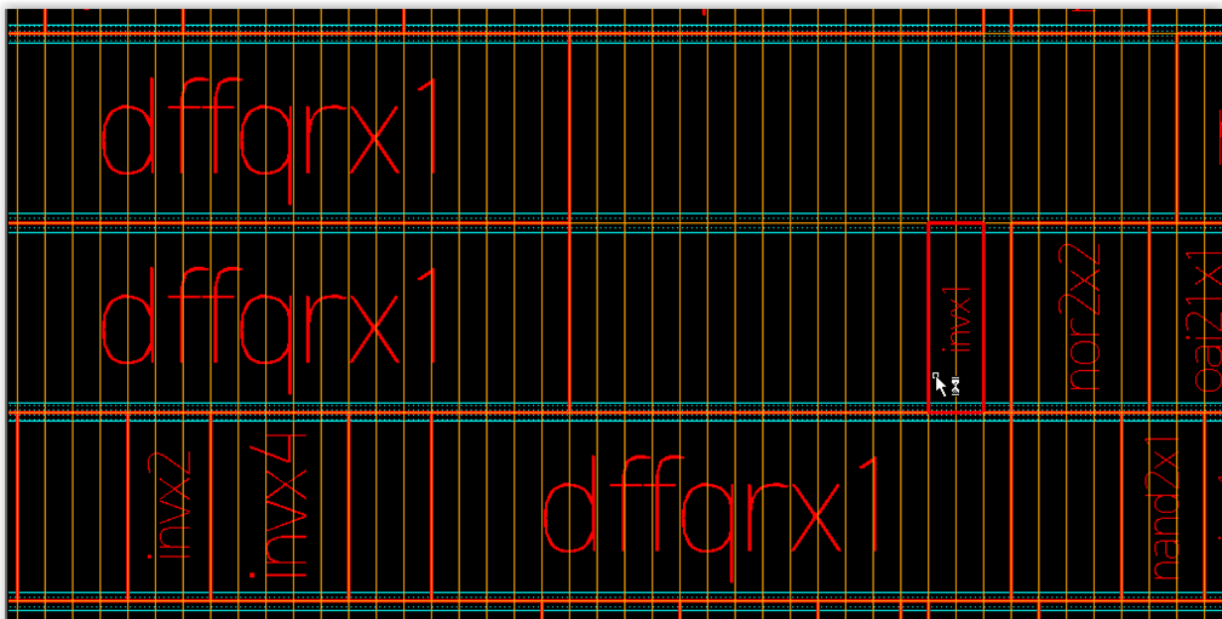
## Adjusting Placement of Cells

You can move standard cells in an IC Compiler design and the tool snaps the standard cells to the rows in the design, at proper multiple locations of the unit tile.

Figure 1, Figure 2, and Figure 3 illustrate the move and snap to row/site locations.



*Figure 1    Selecting the Instance to Move*



*Figure 2    The Instance Snaps to Row Sites during the Move Command*

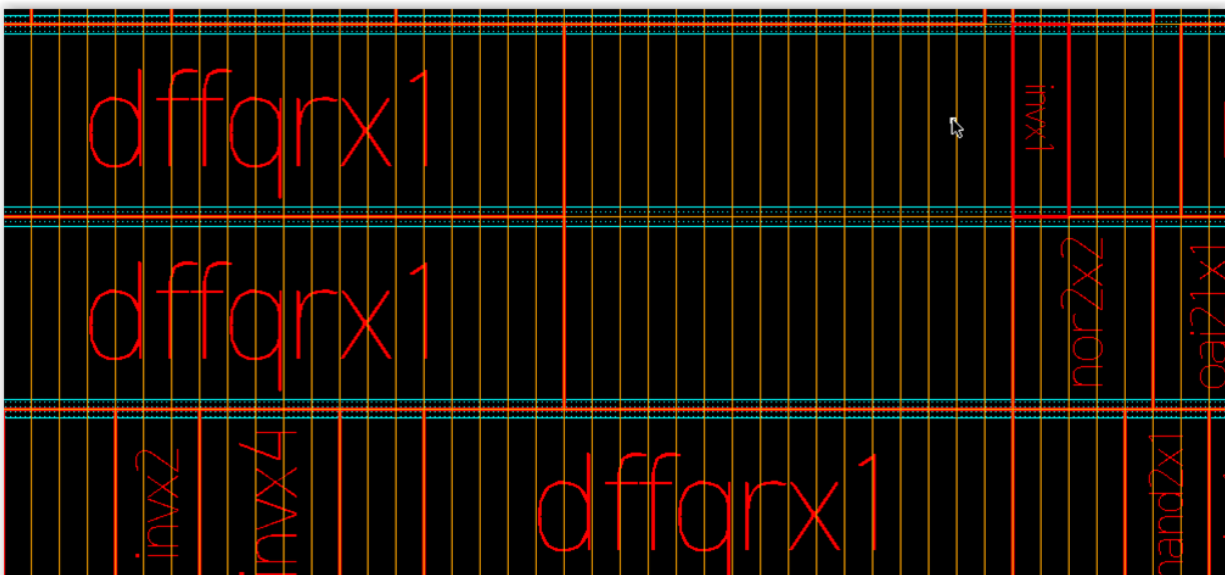


Figure 3 The Instance Orientation Adjusts when Moved to the Next Row in a Double Back Row Configuration

In the figures above, rows are separated by the blue m1 layer power and ground straps. Site locations are identified with the orange lines.

Cells match the required orientation of the row, as shown in [Figure 3](#), where the cell has been flipped (as seen by the location of the beveled corner).

Make sure that cells do not overlap when you move them, because the tool allows cells to be moved over other ones, resulting in overlaps, even though cell overlaps are not legal in IC Compiler.

Once you move cells to the desired location, lock their placement by changing the placement status in the Property Editor assistant.

Once the cell placement status is locked, the tool does not allow moving the cell either manually or automatically in both Custom Designer and IC Compiler.

Macro cells are not bound to respect the row site locations.

---

## Drawing Routes Automatically

You can draw net routes automatically when you are developing analog or mixed signal regions, and you want to allow the Custom Designer Router tool to control the route path.

This is an IC Compiler tool and Custom Designer platform co-design flow.

To draw routes automatically:

1. Begin by adding the IC Compiler design to the Custom Designer environment. See [Designs Originating with the IC Compiler Tool on page 51](#) for details.
2. In the IC Compiler tool, close the design, if it is not yet closed.
3. On the Custom Designer platform, open the design layout view.  
The Custom Designer platform locks the IC Compiler version of this design.  
By default, the Object/Layer Panel assistant opens.
4. Open the Design Navigator assistant by choosing **Window > Assistants > Design Navigator**.
5. Using the Design Navigator assistant, locate and select a net.  
Before drawing net routes, you can optionally use the Design Navigator assistant to create constraints, and set flight line colors. For details, see the [Design Navigator Assistant on page 202](#).
6. Add the Galaxy Custom Router (GCR) menu to the design window by choosing **Tools > Galaxy Custom Router**.
7. Automatically draw the selected net route.
  - a. Choose **Galaxy Custom Router > Auto Route**.
  - b. On the **Auto Route** toolbar, choose the **Selected** option to route the selected net.
  - c. (Optional) Choose the **Shadow** icon in the Design Navigator assistant to dim the design surrounding the selected net.
  - d. Run the router (click the green check mark).
8. View the results. With shadow mode enabled, the net routes appear visually brighter than the surrounding design.
9. (Optional) Save the design at any point.

Saving the design commits all changes to the Milkyway database.

From this point, you can optionally open the design in the IC Compiler tool, perform the next place and route operation, and then optionally return the design to the Custom Designer platform to continue designing.

Before you can open the design for writing in IC Compiler, it has to be either closed or changed to read-only in Custom Designer.

See also

[Drawing Routes Interactively on page 65](#)

More about [Editing an IC Compiler Design in a Custom Designer Environment on page 55](#)

---

## Drawing Routes Interactively

You can draw net routes interactively when you are developing analog or mixed signal regions, and you control the route path.

This is an IC Compiler tool and Custom Designer platform Co-Design flow.

To interactively route designs:

1. Begin by [Preparing Libraries for Co-Design on page 49](#).
2. In the IC Compiler tool, close the design if it is not yet closed.
3. On the Custom Designer platform, open the design layout.
4. Open the Design Navigator assistant by choosing **Window > Assistants > Design Navigator**.
5. Using the Design Navigator assistant, locate, and select a net to route.  
Before drawing net routes, you can optionally use the Design Navigator assistant to create constraints and set flight line colors. For details, see the [Design Navigator Assistant on page 202](#).
6. Activate the Galaxy Custom Router (GCR) tool by choosing **Tools > Galaxy Custom Router**.
7. Interactively draw the route for the selected net by choosing **GCR > Interactive Route** and drawing the route using one of the two modes that follow.

### Follow-the-Cursor Mode

- a. On the **Route** toolbar, deselect the default **Point to Point** option to enable follow-the-cursor mode.
- b. Specify **Multilayer** or **Single Layer**.
- c. On the canvas, click the source pin and draw the route, using the dynamically displayed flight line as a guide to the destination pin.

- d. To complete the path, optionally move the cursor directly over the destination pin or right-click the canvas near the destination pin, and choose **Finish Route**.

**Point-to-Point Mode**

- a. On the **Route** toolbar, choose the default **Point to Point** option, if it is not now selected.
- b. Specify **Multilayer** or **Single Layer**.
- c. On the canvas, click the pin, and then draw the route, clicking again in locations where you want to set intermediate route points.

The router finds the best path along the intermediate route points.

- d. (Optional) Adjust constraint values, and redraw the route.

Use the Constraint Editor assistant to adjust constraints.

- 8. View the results. With shadow mode enabled, the routed nets appear visually brighter than the surrounding design.
- 9. (Optional) Save the design at any point.

Saving the design commits all changes to the Milkyway database.

From this point, you can optionally open the design in the IC Compiler tool, perform the next place and route operation, and then optionally return the design to the Custom Designer platform to continue designing.

Before you can open the design for writing in IC Compiler, it has to be either closed or changed to read-only in Custom Designer.

See also

More about [Editing an IC Compiler Design in a Custom Designer Environment on page 55](#)

---

## Performing IC Compiler Operations on Design Layouts

After you run place and route on a design schematic to implement the block, you can continue to use the IC Compiler tool to work with the block.

You can perform IC Compiler operations on the IC Compiler design (opened in the Custom Designer Layout Editor) using Tcl scripts, or using Console command input that executes IC Compiler commands. Console command input can include both individual IC Compiler commands and Tcl scripts.

These sections cover:

- [Running the IC Compiler Tool using Tcl Scripts for Input on page 67](#)
- [Running the IC Compiler Tool using Console Commands on page 68](#)
- [Setting IC Compiler Bind Keys in Custom Designer on page 68](#)

---

## Running the IC Compiler Tool using Tcl Scripts for Input

You can run IC Compiler tool operations on design layouts and use a Tcl script to determine the input settings.

To run IC Compiler using a Tcl script:

1. On the Custom Designer platform, from the Layout Editor, open the IC Compiler design.

2. In the Layout Editor, choose **ICC > Run ICC**.

This menu option is available only when the IC Compiler design is open.

3. Browse to select the Tcl script file for the system to use.

By default, the tool looks in the current working directory first, unless the `xtICCScript` preference sets it to a different directory location.

4. Click **Open**.

The system evaluates the script in the IC Compiler tool.

Once you choose Apply or OK in the dialog, Custom Designer saves the current design, makes it read-only, and launches IC Compiler with the IC Compiler-selected script. Then, upon completion of the IC Compiler run, the design is refreshed in Custom Designer.

See [Tracking Progress and Viewing Output Files for IC Compiler Operations on page 95](#) to continue.

See also

More about [Editing an IC Compiler Design in a Custom Designer Environment on page 55](#)

---

## Running the IC Compiler Tool using Console Commands

You can run IC Compiler tool operations on design layouts using command input from the Custom Designer Console.

This example executes the IC Compiler `route` command on the active design, using current settings:

```
xt::execICC route_zrt_auto -cellView [de::getActiveContext]
```

This example executes the script `myScript.tcl` and passes it as input to the IC Compiler tool. In the example, `myScript.tcl` is a script you create containing IC Compiler commands.

```
xt::execICC {source myScript.tcl}
```

See the *Custom Designer Tcl Reference Manual* for command descriptions.

When you run the Tcl script, Custom Designer saves the current design, makes it read-only, and launches IC Compiler with the IC Compiler-selected script. Then, upon completion of the IC Compiler run, the design is refreshed in Custom Designer.

See [Tracking Progress and Viewing Output Files for IC Compiler Operations on page 95](#) to continue.

See also

More about [Editing an IC Compiler Design in a Custom Designer Environment on page 55](#)

---

## Setting IC Compiler Bind Keys in Custom Designer

You can switch between IC Compiler and Custom Designer keyboard shortcuts. From the console window, choose **Options > General**. In the dialog, select IC Compiler as the **Active Binding Set**.



## Chapter 5: Editing an IC Compiler Design in a Custom Designer Environment

### Performing IC Compiler Operations on Design Layouts

Not all IC Compiler keyboard shortcuts are supported. Refer to the table of bind keys for the available IC Compiler set below.

<b>IC Compiler Function</b>	<b>IC Compiler Bind Key</b>	<b>Custom Designer Bind Key</b>
Edit > Copy	C	Edit > Copy
Edit > Create > Text	Shift+T	Create > text > Label
Edit > Create > Via	Shift+C	Create > Via
Edit > Delete	D	Edit > Delete
Edit > Merge Net Shapes	Shift+M	Edit > Merge
Edit > Move/Resize	M	Edit > Move
Edit > Properties	Ctrl+R	Property Editor Assistant
Edit > Redo	Ctrl+Y	Edit > Redo
Edit > Stretch Wire	Shift+W	Edit > Stretch
Edit > Undo	Ctrl+Z	Edit > Undo
Highlight > Clear All	Ctrl+M	Highlight > Clear
Select > By Name	Ctrl+B	Ctrl+B
Select > Clear	Ctrl+D	Ctrl+D
Verification > Previous Error	Shift+E	View > Zoom > In
View > Mouse Tools > Ruler Tool	Ctrl+U	Create > Ruler
View > Mouse Tools > Smart Selection Mode	P	Edit > Select > By Region
View > Zoom > Zoom Fit All	F	View > Fit > view
View > Zoom > Zoom Fit Selection	Ctrl+T	View > Fit > Selected

## Chapter 5: Editing an IC Compiler Design in a Custom Designer Environment

### Performing IC Compiler Operations on Design Layouts

IC Compiler Function	IC Compiler Bind Key	Custom Designer Bind Key
View > Zoom > Zoom In	+	View > Zoom > In
View > Zoom > Zoom In	=	NA
View > Zoom > Zoom Out	-	View > Zoom > Out
View > Zoom > Zoom Out	Shift+Z	Shift+Z
View > Zoom > Zoom To	T	View > pan
Window > Close View	Ctrl+W	Design > Close
Remove point during editing command.	Backspace	Backspace
Abort the current active command.	Esc	Esc
In the console window, move backward a character.	Left	Left
In the console window, move forward a character.	Right	Right
In the console window, move to the start of the current line.	Home	Home
In the console window, move to the end of the current line.	End	End
In the console window, select next event in history.	Up	Up
In the console window, select previous event in history.	Down	Down
Set the selectability and visibility of layer on metal1 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+1	Ctrl+1
Set the selectability and visibility of layer on metal2 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+2	Ctrl+2

## Chapter 5: Editing an IC Compiler Design in a Custom Designer Environment

### Performing IC Compiler Operations on Design Layouts

IC Compiler Function	IC Compiler Bind Key	Custom Designer Bind Key
Set the selectability and visibility of layer on metal3 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+3	Ctrl+3
Set the selectability and visibility of layer on metal4 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+4	Ctrl+4
Set the selectability and visibility of layer on metal5 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+5	Ctrl+5
Set the selectability and visibility of layer on metal6 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+6	Ctrl+6
Set the selectability and visibility of layer on metal7 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+7	Ctrl+7
Set the selectability and visibility of layer on metal8 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+8	Ctrl+8
Set the selectability and visibility of layer on metal9 purpose drawing and turn off selectability/visibility for all the other LPPs.	Ctrl+9	Ctrl+9
Toggle selectability and visibility of LPP metal1 drawing.	1	1
Toggle selectability and visibility of LPP metal2 drawing.	2	2
Toggle selectability and visibility of LPP metal3 drawing.	3	3
Toggle selectability and visibility of LPP metal4 drawing.	4	4
Toggle selectability and visibility of LPP metal5 drawing.	5	5

IC Compiler Function	IC Compiler Bind Key	Custom Designer Bind Key
Toggle selectability and visibility of LPP metal6 drawing.	6	6
Toggle selectability and visibility of LPP metal7 drawing.	7	7
Toggle selectability and visibility of LPP metal8 drawing.	8	8
Toggle selectability and visibility of LPP metal9 drawing.	9	9

---

## Use Model Examples

The following sections describe common use models for ICC-CD co-design.

---

### Creating Analog Pre-Routes Targeting a Specific Resistance Value

You might have to implement analog signal nets or analog power nets that target specific resistance values to connect high-speed IP blocks. Custom Designer can help you with such implementation by providing key features such as the Resistance Checker and the Stripe Wire commands.

First, estimate which layers to use with what width to achieve the resistance target for your critical net. Layer and width constraints can be set for that net in the Constraint Editor.

To create the wires that achieve your estimated resistance, use **Create > Interconnect**, **GCR > Interactive Route**, and **GCR > Auto Route**. You can use the Resistance Checker to confirm that the resistance of the route that you created matches the resistance constraint set for that net using the Constraint Editor. If needed, widen the segments of the route to decrease the overall effective resistance. To iterate on the net implementation, you can delete all the shapes on the net using the Context Sensitive Menu (CSM) on the desired net in the Design Navigator. (For more details on routing commands, see the *Custom Designer Custom Router User Guide*. For more details on the

Resistance Checker, see the section "Checking Electromigration and Resistance" in Chapter 12 of the *Galaxy Custom Designer Platform User Guide*.)

The route width might end up being larger than the foundry maximum width process rule for the layers used for the routes. The resistance checker assumes that the net route will be split in stripes and takes that into consideration for calculating the resistance value. You then need to use the **Stripe Wire** command (**Edit > Stripe Wire** from the layout window), which will automatically split the fat wire into multiple parallel stripes. Stripe wire objects have stripe attributes that can be modified in the attribute dialog. (For more details on **Stripe Wire**, see Chapter 7 of the *Galaxy Custom Designer Layout Editor User Guide*.)

---

## Cleaning Up the Layout after Routing

Once routing is completed in the design, whether partially or fully, you can run a sign off physical verification tool to validate the layout compliancy with foundry rules. This is a required step before sending the design to the foundry.

Custom Designer has interfaces to several physical verification tools, such as IC Validator.

Once the physical verification job is set up and executed, the debugging environment for the same tool (ICV VUE for instance) is launched, in case there are violations to review and fix.

The debugging window can be used to cross-probe to specific locations in the layout and highlight the violations.

Custom Designer includes an advanced feature to help fix violations, DRD Autofix, which you can access using **Edit > SmartDRD Autofix** from the layout window. By drawing a region over the area that has the violations, DRD Autofix detects the potential violations and attempts to fix them with minimal changes to the layout.

You can also use the whole range of Custom Designer manual editing capabilities to correct the DRC.

## Working with Routing Objects in an ICC-CD Flow

When using Paths or Rectangles as routing objects, you might encounter some issues. Therefore, you should use PathSeg as the routing object for the ICC-CD flow using Galaxy Custom Router (GCR) applications whenever possible.

Custom Designer supports an option to choose Path or PathSeg as the routing objects for various route creation commands such as **Create > Interconnect** and **GCR > Interactive Route**.

To set Path or PathSeg objects, choose **Options > Design** in the layout window to bring up the **Layout Design Options** dialog (Figure 4).

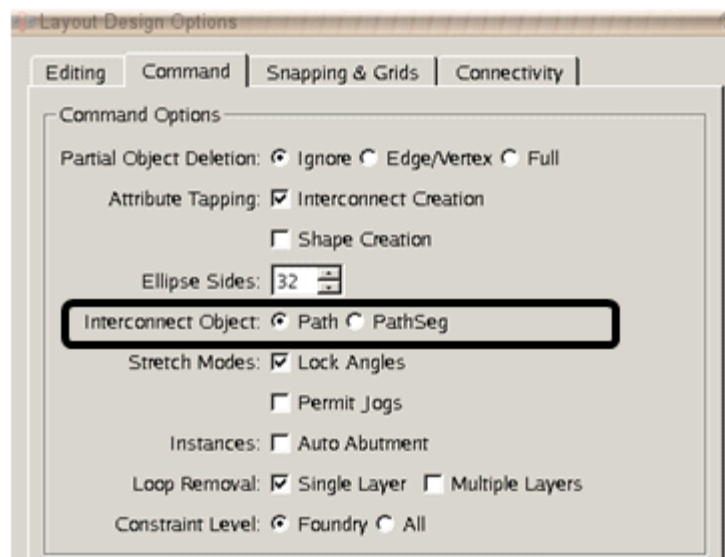


Figure 4 Setting Path/PathSeg

In the **Command** tab, choose **Path** as the **Interconnect Object** (shown in Figure 4).

## Paths

Note that when Path is used as the routing object, you might encounter the following issues:

- *Known Issue: False DRC violations*

For nets in an IC Compiler design, shielding is performed in Custom Designer and the design is verified in IC Compiler using `verify_zrt_route -check_from_user shapes true`. The tool flags "Diff net var rule" spacing violations between shield and signal routing.

These violations are false, because IC Compiler does not honor shield attributes on the Path objects.

Solution: Use PathSeg as the routing object to avoid false DRC violations.

- *Known Issue: Adding buffer on long routes*

The IC Compiler `add_buffer_on_route` command does not support routes created using Path objects.

Solution: Use PathSeg as the routing object.

## Rectangles

Rectangles might be created on the design to address various design needs, such as fixing DRC errors. However, IC Compiler does not support rectangles as part of routing, so it is necessary to convert Rectangles to Wires before routing. This can help avoid issues in later ICC applications and commands; for example, if the nets associated with routing have to undergo an ECO flow in IC Compiler, the IC Compiler command `remove_net` will fail if there are Rectangle objects.

To convert Rectangles to Wires, use the script found in SolvNet article #030593, ["Script to Convert Rectangle Shapes to Wires."](#)

## **Chapter 5: Editing an IC Compiler Design in a Custom Designer Environment**

### Use Model Examples



## Digital Timing Shells for Analog Blocks

---

*How to create digital timing shells for analog blocks to export them to IC Compiler.*

You can create timing shells for running IC Compiler place and route. Digital timing shells are created for analog blocks to export the analog block to IC Compiler with a timing model to use the block in the next level of hierarchy for top-level implementation in IC Compiler. Using timing shell methodology, an analog block is wrapped with a thin digital logic. The advantages to this method are that

- no layout is required
- analog can be a black box
- no SPICE simulation needed
- covers all operating conditions
- automated support for multi-voltage
- model view is not dependent on analog cell changes

IC Compiler is used for placement and routing of the timing shell. This includes placement of standard cells, black boxes (size and floor plan constraints are defined in Custom Designer), and pins. The output of the place-and-route flow is a placed-and-routed Milkyway representation of the timing shell, with a power/ground network created using UPF annotation.

A block abstraction can also be generated for the timing shell. This block abstraction is the timing model and helps to perform an accurate timing analysis at the next level up in the hierarchy where the block will be instantiated. The black box can then be edited in Custom Designer to perform the layout of the analog content.

These sections cover:

- [Timing Shells on page 78](#)
- [Creating Timing Shells for Analog Blocks on page 80](#)

---

## Timing Shells

A timing shell represents a mixed-signal design with one or more analog blocks (black boxes) and a semi-custom digital shell wrapped around those full-custom blocks. A digital timing shell mainly consists of flip-flops (see [Figure 1](#)), which enables creation of a block abstraction in IC Compiler. At the chip level, IC Compiler performs an accurate timing analysis using the block abstraction of the timing shell block.

# Timing Shell Structure

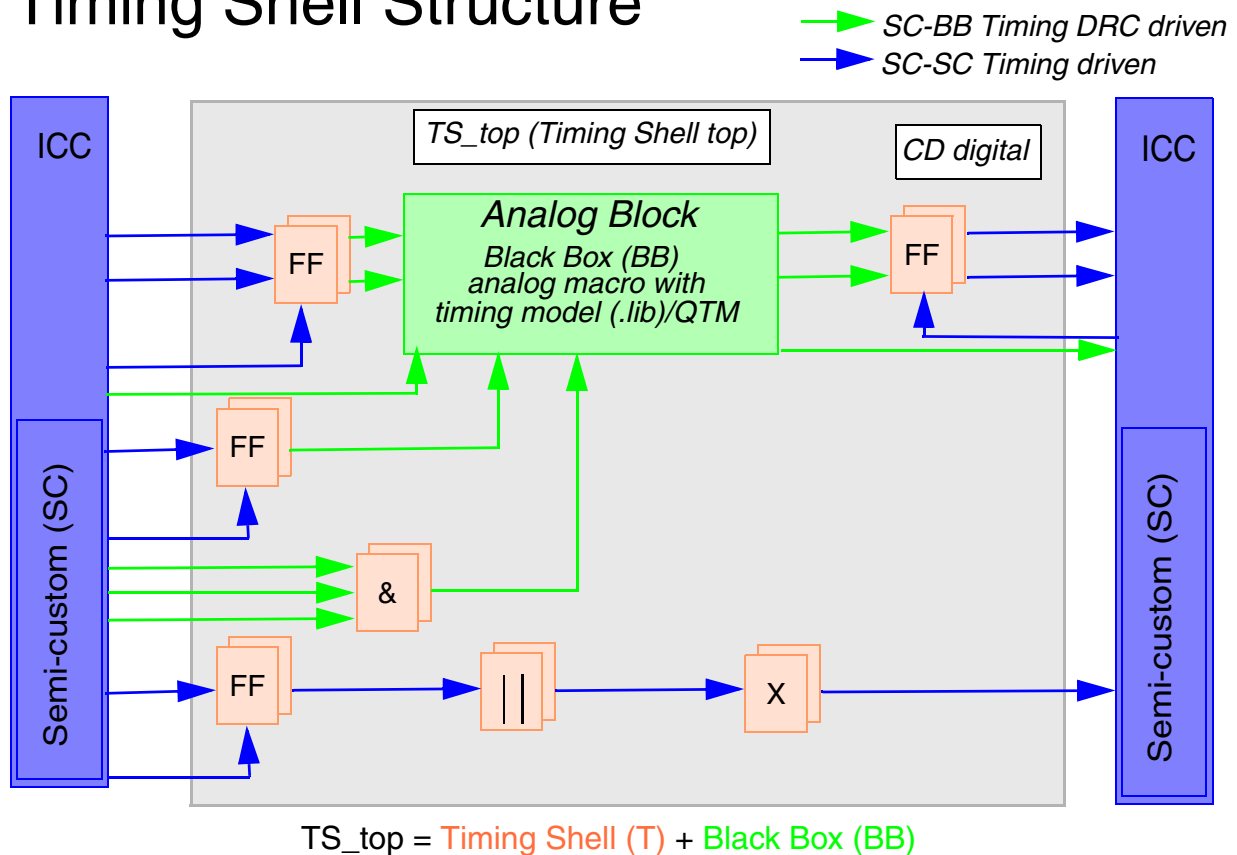


Figure 1 Timing Shell Structure

In Figure 1, the green box contains the analog block within a black box. The gray region is the digital timing shell, which can contain:

- semi-custom components (e.g. gates, interconnects, pins)
- instances of black boxes (BB)

The digital timing shell cannot contain non-semi-custom components, such as pcells.

All timing paths end in the timing shell in registered in- and outputs. No timing path can end on black-box pins (design-rule driven only).

See also

More about [Digital Timing Shells for Analog Blocks on page 77](#)

---

## Creating Timing Shells for Analog Blocks

Before you perform place and route for digital blocks with the IC Compiler tool, you can create digital timing shells for analog blocks. The analog block is treated or defined as a black box during place and route. Only the analog block's power, ground, and signal pin connections interact with the digital design during place and route.

Define black boxes by calculating their dimensions and specifying pin connections for the power and ground pins that will connect the analog block with the surrounding design. You can define black boxes with single power and ground connections, or UPF multiple power and ground connections.

You can use this procedure both to define a new black box, and to edit an existing black box.

The following sections describe an IC Compiler and Custom Designer Co-Design flow that begins with a design schematic. The flow consists of the following steps:

- [Prepare to Create a Timing Shell on page 80](#)
- [Create Connection Definitions for the Timing Shell on page 82](#)
- [Define the IC Compiler Black Box on page 83](#)

---

### Prepare to Create a Timing Shell

Before you create a timing shell, you might want to create timing libraries and/or modify the Place and Route command file to create a block abstraction. The section [Customizing the IC Compiler Place and Route Command File on page 97](#) describes how to modify the Place and Route command file.

1. (Optional) If you plan to use multiple power supplies using power supply definitions or a UPF file, or you want to create a block abstraction, set up timing libraries.
  - If timing libraries now exist, use the preferences in [Table 1 on page 81](#) to specify them.

- If timing libraries do not exist, you can create them. See [Creating Timing Libraries for Multi-Voltage Power in Place and Route on page 53](#).

[Table 1 on page 81](#) lists and describes preferences for setting up existing timing libraries. For additional assistance in using these preferences, contact your Synopsys Support representative.

*Table 1 Preferences for Setting Up Timing Libraries*

Preference	Description
leICCSearchPath	Sets the value of ICC_SEARCH_PATH in the IC Compiler template file, which is in turn appended to the IC Compiler variable <code>search_path</code> . The IC Compiler tool uses it to resolve relative <code>.db</code> file names in the <code>link/target_library</code> .
leICCLinkLibrary	Tcl list of <code>.db</code> file names. The list should contain the <code>.db</code> files for standard cell libraries used in the design. Sets the <code>ICC_LINK_LIBRARY</code> variable in the IC Compiler template file.
leICCTargetLibrary	Tcl list of <code>.db</code> file names. The list should contain the <code>.db</code> files for standard cell libraries used in the design. Sets the <code>ICC_TARGET_LIBRARY</code> variable in the IC Compiler template file.

2. (Optional) If you want the system to create a block abstraction after place and route, set the `ICC_CREATE_TIMING_MODEL` variable to true in the IC Compiler Place and Route command file.

Then specify the values for the `TLUplus` files.

[Table 2 on page 81](#) lists and describes the preferences you need to use to set up block abstractions.

*Table 2 Preferences for Setting up Block Abstraction*

Preference	Description
leICCTLUPlusMapFile	Sets the value of the <code>TLUPLUS_MAP_FILE</code> in the IC Compiler template. Specifies the name of the file that defines the layer mapping between the IC Compiler technology file and the Interconnect Technology Format (ITF) file.

*Table 2 Preferences for Setting up Block Abstraction*

Preference	Description
leICCTLUPlusMinFile	Sets the value of the TLUPLUS_MIN_FILE in the IC Compiler template. The IC Compiler tool uses this TLUPlus file for minimum condition calculation of resistance and capacitance.
leICCTLUPlusMaxFile	Sets the value of the TLUPLUS_MAX_FILE in the IC Compiler template. The IC Compiler tool uses this for maximum condition calculation of resistance and capacitance.

See also

More about [Digital Timing Shells for Analog Blocks on page 77](#)

---

## Create Connection Definitions for the Timing Shell

Connection definitions enable the use of IP from different sources using multiple power supplies by mapping to power domains and connecting supply statements. For a multi-power-supply flow that uses UPF file generation, the schematic needs to have connection definitions.

After preparing your timing libraries and creating the necessary views by modifying the IC Compiler Place and Route command file, you can create connection definitions.

1. Open the design schematic you want to use.
2. In the Schematic Editor, choose **Tools > ICC Link**.
3. Follow the instructions for [Creating Connection Definitions](#) in the *Schematic Editor User Guide*.

See [Table 3](#) for details about connection definition mapping.

*Table 3 How Connection Definitions Map to UPF Power Domains and Supply Nets*

Connection Definitions	UPF Mapping	Example
All connection definitions	Supply nets/ports are created Note: ! is removed from the names	<pre>create_supply_port vdd create_supply_net vdd connect_supply_net vdd -ports vdd</pre>
stdcells connection definitions	Primary Power net/ Ground net Top Power Domain is created with these nets	<pre>set_domain_supply_net Top_PD -primary_power_net vdd -primary_ground_net vss</pre>
BlackBox with connection definitions	Supply nets are created and black box ports are connected to the supply nets	<pre>create_supply_port vdda create_supply_net vdda connect_supply_net vdd -ports vdda connect_supply_net vdda -ports {BB1/vdda}</pre>

See also

More about [Digital Timing Shells for Analog Blocks on page 77](#)

## Define the IC Compiler Black Box

Optionally, you can create an IC Compiler black box. The black box is intended for use with an analog instance whose layout is not available yet. The timing shell also applies to a cell that has an analog cell and a digital interface around it.

**Note:** Currently the flow does not support UPF generation for an analog cell that already has a completed layout.

1. (Optional.) Choose **ICC > Define BlackBox**, and select the instance you want to use.
2. In the **Create ICC Black Box** dialog box, begin to define the black box as described in [Table 4 on page 84](#).

If you are editing an existing black box, the dialog box title shown is **Edit ICC Black Box**.

[Table 4 on page 84](#) lists and describes the settings required to create or edit an ICC black box.

*Table 4 Create or Edit ICC Blackbox (top section of the dialog box)*

Setting	Description
<b>Library</b>	The library where the design resides, from the selected instance.
<b>Cell</b>	BlackBox (default), from the selected instance.
<b>View</b>	blackbox (default)
<b>Technology</b>	Adds technology details that supply layer information, for example, layer constraints for signal pins, and the layer used for power/ground pin creation. The system also uses the technology to determine the <code>minWidth</code> of a layer when you specify the <b>Use Default (3x minWidth)</b> setting in the Add/Edit pin dialog box.
<b>Size</b> (Option provided by the IC Compiler tool)	<p><b>Width/Height</b> – Calculates the black box dimensions in microns from the width and height you enter here. For example: 10 x 10. You can use this option when you want the black box to be a rectangle and you know the size.</p> <p><b>Gate Count</b> – Calculates the black box dimensions from the <b>Gate Count</b> and <b>Utilization</b> you enter here. You can use this option if you know the approximate size in numbers of gates.</p> <p><b>Boundary</b> – Calculates the black box dimensions, in microns, from the coordinates you enter here. For example: {{0 0} {0 200} {230 200} {230 0}}. You can use this option when you have a non-rectangular boundary already defined in the layout.</p>

3. In the **Create ICC Blackbox** dialog box under **Pins**, continue by designating supply pins.



- If you are creating a new blackbox cellView, the signal pins are copied from the placed master of the selected instance. The power/ground pins are copied from the inherited connections of the schematic hierarchy below the selected instance.
- If you are editing an existing blackbox definition, the pins are loaded from the existing blackbox cellView. If the schematic design does not contain a full schematic implementation of the blackbox cellView, you might need to explicitly add power/ground pins (see the step that follows).

Alternatively, you can copy the pins from an existing cellView. To copy pins, click **Extract**, enter details about the source view as described in [Table 5 on page 85](#), and click **OK**.

Table 5 Extract and Copy Pins for a Black Box (middle section of the dialog box)

Setting	Description
<b>Library</b>	The library that contains the source design view.
<b>Cell</b>	The cell of the source design view.
<b>View</b>	The view of the source design.
<b>Hierarchy</b>	<p><b>View Search List</b> – Defines the list of views to search to find the master of the source design. Separate each view name with a space.</p> <p><b>Stop List</b> – Defines the view that stops the search where the power supply is defined. This view must have either a terminal or net connection definition.</p>

4. In the **Create ICC Blackbox** dialog box under **Pins**, optionally add, edit, and remove power and ground pins.
  - To add pins, click **Add**, enter details as described in [Table 6 on page 86](#), and click **OK**.
  - To edit details for a power or ground pin in the list, select the pin name, click **Edit**, adjust details as described in [Table 6 on page 86](#), and click **OK**.

## Chapter 6: Digital Timing Shells for Analog Blocks

### Creating Timing Shells for Analog Blocks

- To remove a pin from the list, select it by name in the list, and then choose **Remove**.

*Table 6 Add or Edit Pin Connections for a Black Box (middle section of the dialog box)*

Setting	Description
<b>Name</b>	The name of the pin.
<b>Type</b>	<b>Power</b> – Specifies that this pin is a power pin. <b>Ground</b> – Specifies that this pin is a ground pin. <b>Layer</b> – The one you choose from the alphanumerically sorted list of metal layers in the drop down menu.
<b>Pin Size</b>	<b>Use Default (3x minWidth)</b> – Applies the default size shown. <b>Width</b> – Applies a non-default size to the pin with the width you specify. <b>Height</b> – Applies a non-default size to the pin with the height you specify.
<b>Connection Assignment</b>	<b>Name</b> – The name of the connection definition created for the new terminal. <b>Default Net</b> – The name of the default net for the connection definition.

5. In the **Create ICC Blackbox** dialog box under **Pin Constraints**, complete the black box definition as described in [Table 7 on page 86](#).

*Table 7 Create or Edit ICC Blackbox (lower section of the dialog box)*

Setting	Description
<b>Pin Constraints</b>	<b>Allowed Layers</b> – The layers you want the IC Compiler tool to use for creating signal pins. The system finds these in the technology library. Layers listed here correspond to the alphabetically sorted list of all OpenAccess physical layers for which oaMaterial=metal.

Table 7 Create or Edit ICC Blackbox (lower section of the dialog box)

Setting	Description
	<b>Preroute Spacing</b> – Designates the spacing between pins in the black box. Can be 0 or greater.
	<b>No Stacking</b> – Controls signal pin stacking. Options include no stacking for <b>all_pins</b> , stacking for <b>pg_pins_only</b> or <b>signal_pins_only</b> , or <b>stacking_allowed</b> .
	<b>Bus Ordering</b> – Designates the bus order for the IC Compiler tool to recognize. Choices are <b>lsb_to_msb</b> , <b>msb_to_lsb</b> , <b>scrambled</b> , or <b>consistent_wirelengths</b> .
	<b>Keep Buses Together</b> – Specifies that buses be kept together

6. In the **Create ICC Blackbox** dialog box that shows details for the black box you have defined, click **OK**.

The system creates the black box view with this definition.

If you edited a black box view, the system updates its definition.

7. Run Place and Route.

Continue by [Developing Digital Blocks with Co-Design on page 89](#).

See also

More about [Digital Timing Shells for Analog Blocks on page 77](#)

## **Chapter 6: Digital Timing Shells for Analog Blocks**

### Creating Timing Shells for Analog Blocks

## Developing Digital Blocks with Co-Design

---

*How to develop digital blocks.*

You can implement and develop digital blocks in your mixed signal designs. To do this, use the IC Compiler tool to place and route the blocks through the tool's interface with the Custom Designer platform.

You can optionally choose to create a block abstraction during the place and route process.

You can create a custom process flow with these additional procedures:

- You can add floorplanning conditions when you want to cause the IC Compiler tool to recognize and respect them during place and route. See [Adding Floorplanning Conditions for Digital Blocks on page 90](#).
- After you run place and route, you can customize IC Compiler processing by editing the IC Compiler command file. [Customizing the IC Compiler Place and Route Command File on page 97](#).

These sections cover:

- [Adding Floorplanning Conditions for Digital Blocks on page 90](#)
- [Implementing Digital Blocks on page 90](#)
- [Customizing the IC Compiler Place and Route Command File on page 97](#)

## Adding Floorplanning Conditions for Digital Blocks

From the Custom Designer platform, you can perform IC Compiler place and route using the created layout with floorplanning conditions as a template.

This is an IC Compiler tool and Custom Designer platform Co-Design flow.

To add floorplanning conditions:

1. On the Custom Designer platform, open the design layout.
2. Create floorplanning conditions.

Floorplanning conditions can include, for example, rectilinear boundaries to physically constrain the digital block placement area, straps for pre-routing the instances, and pin placement to constrain pin locations. Straps are paths or path segments, which can be connected to power or ground.

3. From the Custom Designer platform, perform IC Compiler place and route. See [Implementing Digital Blocks on page 90](#).

See also

More about [Developing Digital Blocks with Co-Design on page 89](#)

---

## Implementing Digital Blocks

You can implement digital blocks by using the IC Compiler tool to run place and route, and to perform other operations on mixed signal designs.

These sections cover:

- [Running IC Compiler Place and Route from Design Schematics or Verilog Netlists on page 91](#)
- [Tracking Progress and Viewing Output Files for IC Compiler Operations on page 95](#)
- [Designing with Place and Route Results on page 96](#)

## Running IC Compiler Place and Route from Design Schematics or Verilog Netlists

You can run place and route with the IC Compiler tool using either a design schematic or a Verilog netlist for the design source.

This process produces a new block based on the schematic, and uses, when available, layout shapes to define floorplanning conditions, as imported from the design layout.

Prepare for place and route by [Preparing Libraries for Co-Design on page 49](#).

If you want to specify multiple or UPF power supplies, create a timing shell before running place and route. (See [Creating Timing Shells for Analog Blocks on page 80](#).)

If you want to include floorplanning conditions, you can add these to your design layout before running place and route. (See [Adding Floorplanning Conditions for Digital Blocks on page 90](#).)

To run place and route:

1. On the Custom Designer platform, open the schematic view of the design.
2. Initiate the IC Compiler place and route process:
  - a. Choose **Tools > ICC Link**.
  - b. Choose **ICC > Place and Route**. The **Place & Route with ICC** dialog opens ([Figure 1](#)).

## Chapter 7: Developing Digital Blocks with Co-Design

### Implementing Digital Blocks

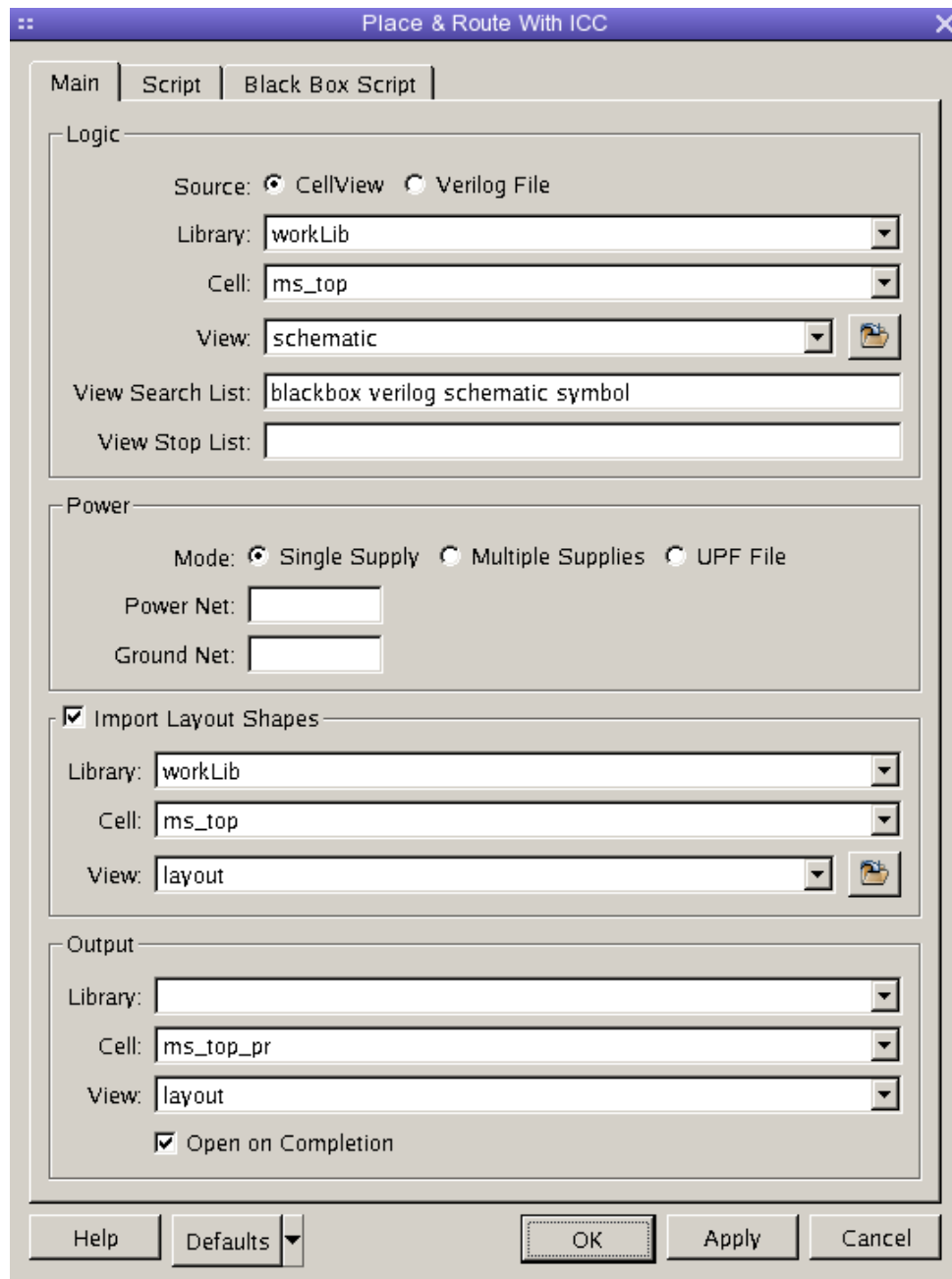


Figure 1 Place & Route with ICC Dialog

3. Enter settings on the **Main** tab.



[Table 1 on page 93](#) lists and describes place and route settings.

Table 1 Place & Route with IC Compiler (Main tab)

Settings	Description
<b>Logic</b>	<p>Specify the source type.</p> <p>If you have a <b>CellView</b> source, specify the <b>Library</b>, <b>Cell</b>, and <b>View</b>. Additionally specify a <b>View Search List</b>, and <b>View Stop List</b>, or accept the default lists.</p> <p>If you have a <b>Verilog File</b> source, browse to select the file, and enter the <b>Top Module</b> name.</p>
<b>Power</b>	<p><b>Single Supply</b> – Specifies a single power source with the <b>Power Net</b> and <b>Ground Net</b> you enter here. If your design has a single supply net without a black box, choose this option. If your design has a single supply net, and one or more black boxes, you can choose this option if the black boxes do not have multiple power and ground terminals.</p> <p><b>Multiple Supplies</b> – Extracts the power network information from this design for outputting in UPF format. Running place and route once with multiple supplies generates a UPF file you can use to rerun place and route with unified power format.</p> <p>The <b>Multiple Supplies - Domains</b> tab shows the common power and ground connections shared by the gates in the design.</p> <p>The <b>Multiple Supplies - Black Boxes</b> tab shows the power and ground details for any black boxes.</p> <p>The <b>Multiple Supplies - Supply Nets</b> tab shows all power and ground details for the design. If your design has multiple supplies, with or without one or more black boxes, you must choose this option. When your design has inherited connections to define the supplies, you must choose this option to make sure inherited connections for power and ground nets are honored.</p>

*Table 1 Place & Route with IC Compiler (Main tab)*

Settings	Description
	<b>UPF File</b> – Uses the Unified Power Format (UPF) file you browse for and select here to define power and ground settings.
<b>Import Layout Shapes</b>	Specifies a <b>Library</b> , <b>Cell</b> , and <b>View</b> to use as a layout template for layout shapes. This setting is enabled by default, with the current design layout used as the template. See <a href="#">Adding Floorplanning Conditions for Digital Blocks on page 90</a> .
<b>Output</b>	Specifies the IC Compiler target <b>Library</b> location for running the place and route process, and the <b>Cell</b> and <b>View</b> to produce as output. The tool appends the characters <b>_pr</b> to the cell name to prevent a naming conflict with the place and route layout template. If you are rerunning place and route, for example, with a UPF file generated by a previous run with multiple supplies, you can choose to rename the output cellView to retain the results of the previous run.  <b>Open on Completion</b> optionally opens the design after the place and route process completes.

- On the **Script** tab, optionally view the default Tcl script template shipped with the tool, `icc-place_route-template.tcl`.  
If you have created a custom Tcl script, you can browse to select it for this place and route job.
- If this design contains a black box, on the **Black Box script** tab, optionally view the default black box Tcl script template shipped with the tool, `BlackBox.tcl`.
- Click **OK**.  
The tool extracts a Verilog netlist and then performs place and route on the design. See [Tracking Progress and Viewing Output Files for IC Compiler Operations on page 95](#) to continue.

See also

More about [Developing Digital Blocks with Co-Design on page 89](#)

---

## Tracking Progress and Viewing Output Files for IC Compiler Operations

You can track progress, and view job output files, for all IC Compiler operations executed from the Custom Designer platform.

To track progress and view output files:

1. Open the Job Monitor by choosing **Tools > Job Monitor** in the Layout Editor.
2. Select the job by name.
3. Right-click and choose **View Output**.

If you chose to open the design on completion when you set up the job, it opens in the Layout Editor.

If you did not choose to open it on completion, you can open the file `design_pr` to see the place and route results.

4. In the Job Monitor, you can view output files by clicking the tabs.

During place and route, the system creates a custom command file from the command template by applying the current settings. It netlists the design, runs place and route, and produces output files.

The tool stores the output files in the `iccbbridge/place_route` directory by default.

[Table 2 on page 95](#) lists and describes the output files.

*Table 2 IC Compiler Place and Route Output Files*

File	Description
<code>icc-place_route-template.tcl</code>	Supplied template for command file.
<code>icc-place_route.tcl</code>	Tcl script for this run.
<code>icc-place_route.log</code>	Log file for this run.

*Table 2 IC Compiler Place and Route Output Files*

File	Description
<code>&lt;cellName&gt;.v</code>	Verilog netlist for this run. If multiple power supply UPF power output was selected, power and ground net information are not included.
<code>&lt;cellName&gt;.upf</code>	Netlist for multiple power supply output in UPF format for this run. Contains power and ground nets. You can use this file as input for rerunning UPF format place and route.

For timing shell black boxes, the system will create a Tcl file for each black box in the hierarchy. These black box Tcl files communicate boundary and pin constraints and initial pin shapes for power/ground and signal nets to the IC Compiler tool.

If you have a layout with floorplanning conditions set, the IC Compiler tool has recognized them during place and route.

If you defined a black box, the IC Compiler tool honors its boundaries, and recognizes its power and ground connections. To see the power and ground pins, descend into the black box instance.

5. (Optional) View the improved design. See [Designing with Place and Route Results on page 96](#) to continue.

See also

More about [Developing Digital Blocks with Co-Design on page 89](#)

---

## Designing with Place and Route Results

You can design with place and route results by using both the Custom Designer Layout Editor and the IC Compiler tool for viewing, editing, and optionally rerunning the place and route process.

To design with place and route results:

1. In the design layout on the Custom Designer platform, view the initial routing results.

To obtain place and route results, see [Running IC Compiler Place and Route from Design Schematics or Verilog Netlists on page 91](#).

If you chose to open the design on completion when you set up the job, it opens automatically in the Layout Editor.

If you did not choose to open it on completion, you can open the file *design\_pr* to see the place and route results.

2. (Optional) In the Layout Editor, edit the design.
3. When you are ready, choose **ICC > Open in ICC** to view the results in the IC Compiler tool.

**Note:** This menu option is available only when an IC Compiler tool design is opened in LE.

If you edited the design in the Layout Editor, save changes in preparation for allowing the LE version to enter *read-only* mode.

The design opens in an IC Compiler tool window in *write* mode.

4. Continue to develop the design with the IC Compiler tool.  
See the Synopsys IC Compiler tool documentation for instructions.
5. When you are ready, save changes and close the IC Compiler tool.  
The Custom Designer Layout Editor opens the design in *editing* mode, showing the recent changes.
6. Continue to develop the design with the Layout Editor.

See also

More about [Developing Digital Blocks with Co-Design on page 89](#)

---

## Customizing the IC Compiler Place and Route Command File

You can customize the IC Compiler command file to add execution of specific operations during place and route processes you run from the Custom Designer platform.

After running place and route once, you can edit the command file, and run place and route again with custom settings.

For example, where your design has standard cells with instance placement already defined, you can edit the command file to perform route drawing only.

You can also enable IC Compiler to create a FRAM view from the block.

This is an IC Compiler and Custom Designer Co-Design flow.

To use a custom command file to control place and route:

1. Begin by [Running IC Compiler Place and Route from Design Schematics or Verilog Netlists on page 91](#).
2. After running the place and route process once, open the command file at this location: `iccbridge/place_route/icc-place_route.tcl`
3. Copy this command to a file with a new name.
4. Edit the file.

Under the section **Operations to Perform**, set listed operations to true or false.

(Optional) Add IC Compiler commands for additional operations.

5. Save the file in the same directory to make it available from the **Commands** tab in the **Place and Route with ICC** dialog box.

For future place and route processes, you can browse to select the custom file from the **Commands** tab.

See also

More about [Developing Digital Blocks with Co-Design on page 89](#)

## Tutorial: Constraint Rules Mapping

---

*Provides a tutorial introduction to IC Compiler-Custom Designer round-trip flows.*

The design used in this tutorial is a mixed-signal design where the top design has a completed floor plan and initial block placement using the IC Compiler tool. The design contains these design elements:

- a dcc -Duty Cycle Converter, an analog block implemented in Custom Designer using reference40nm PDK
- a I106 - mixed signal block
- a cpu|interrupt - digital block

As a part of the tutorial, you will

- review existing non-default rules in IC Compiler design.
- create new constraints and rules in Custom Designer.
- save the design back to IC Compiler and review the rules in IC Compiler.

---

### Before You Start

Complete the following set-up instructions before you begin this tutorial.

---

## Files

All files for this lab are located in the directory

CDICC\_NonDefaultRules\_Lab.

1. From the CDICC\_NonDefaultRules\_Lab directory, run the `icc_setup` from a UNIX shell to set the correct IC Compiler reference library paths in IC Compiler.
2. Verify that you have the files listed below, in the following directory structure:

---

CDICC_NonDefaultRules_Lab	Current working directory
iccScripts	IC Compiler scripts
chip_mstop_icc_preroute	IC Compiler library
iccRefLibs	IC Compiler Reference Libraries
tech	ICV runsets
tcl	Custom Designer Tcl scripts

---

---

## Product Versions

Use the following product versions for this tutorial:

---

Product	Version
Custom Designer	J-2014.12
IC Compiler	J-2014.09

---

---

## Task 1: Analyze the Design Using IC Compiler

In this task you will analyze the IC Compiler version of the design to understand the objects in the design.

1. In the Unix Window, change your working directory to CDICC\_NonDefaultRules\_Lab.



2. Run the `icc_setup` from a UNIX shell to set the correct IC Compiler reference library paths in IC Compiler:

```
unix% ./icc_setup
```

3. Launch the IC Compiler tool:

```
unix% icc_shell -gui &
```

4. Choose **File > Open Design** and:

- a. Select **chip\_mstop\_icc** for the **Library Name**.

- b. Select **cpu\_preRoute** for **CEL**.

Two non-default rules (**w2\_w1\_side** and **mvcp\_mvcm\_nets**) are defined in the design. Review these rules in the following steps.

5. Choose **Define Routing Rule (Route > Routing Setup > Define Routing Rule)**.

- a. Click on the **w2\_w1\_side** rule.

- b. Observe that layers m2, m3, m4 have non-default rules for width set to 0.1u and shield width and spacing 0.2u defined for all layers, as shown in [Figure 2](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 1: Analyze the Design Using IC Compiler

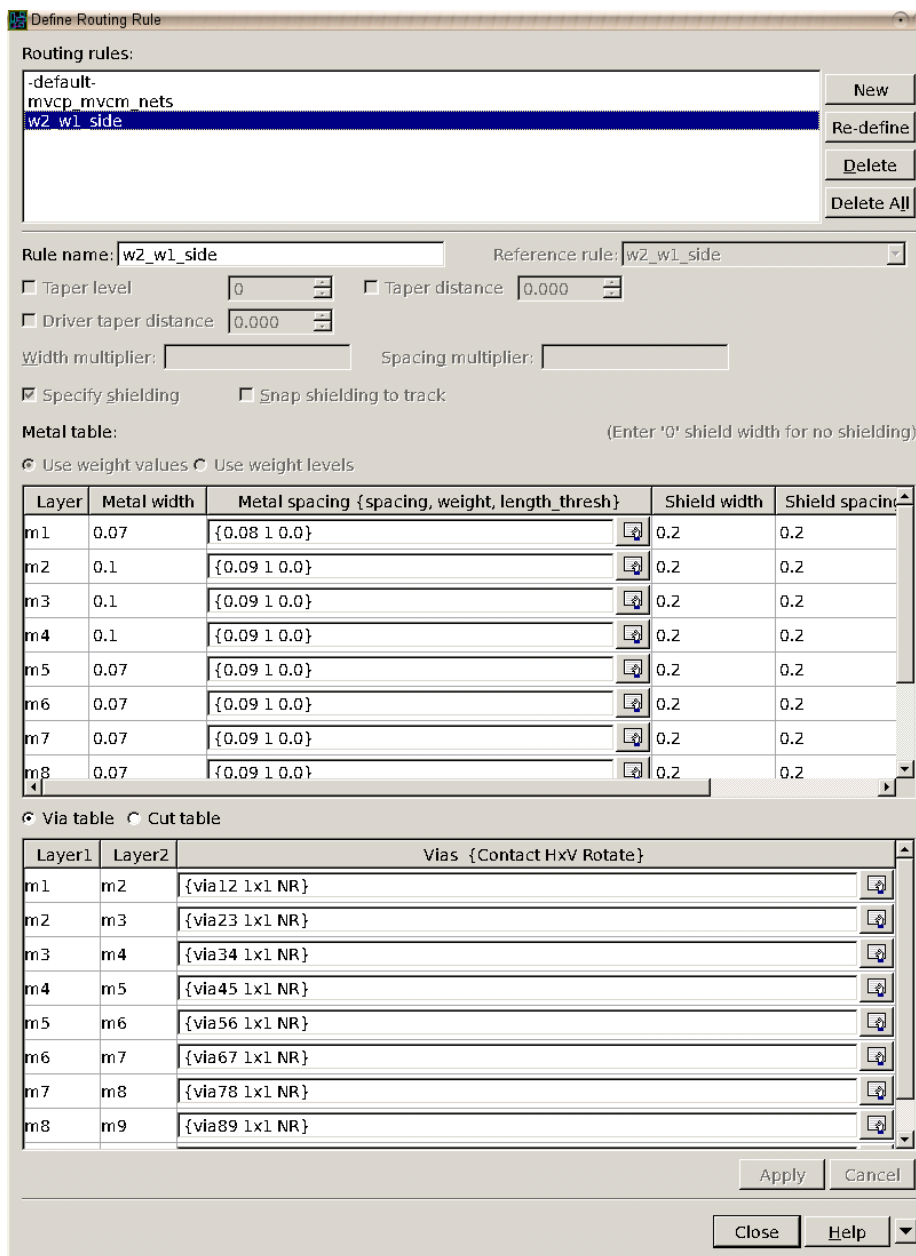
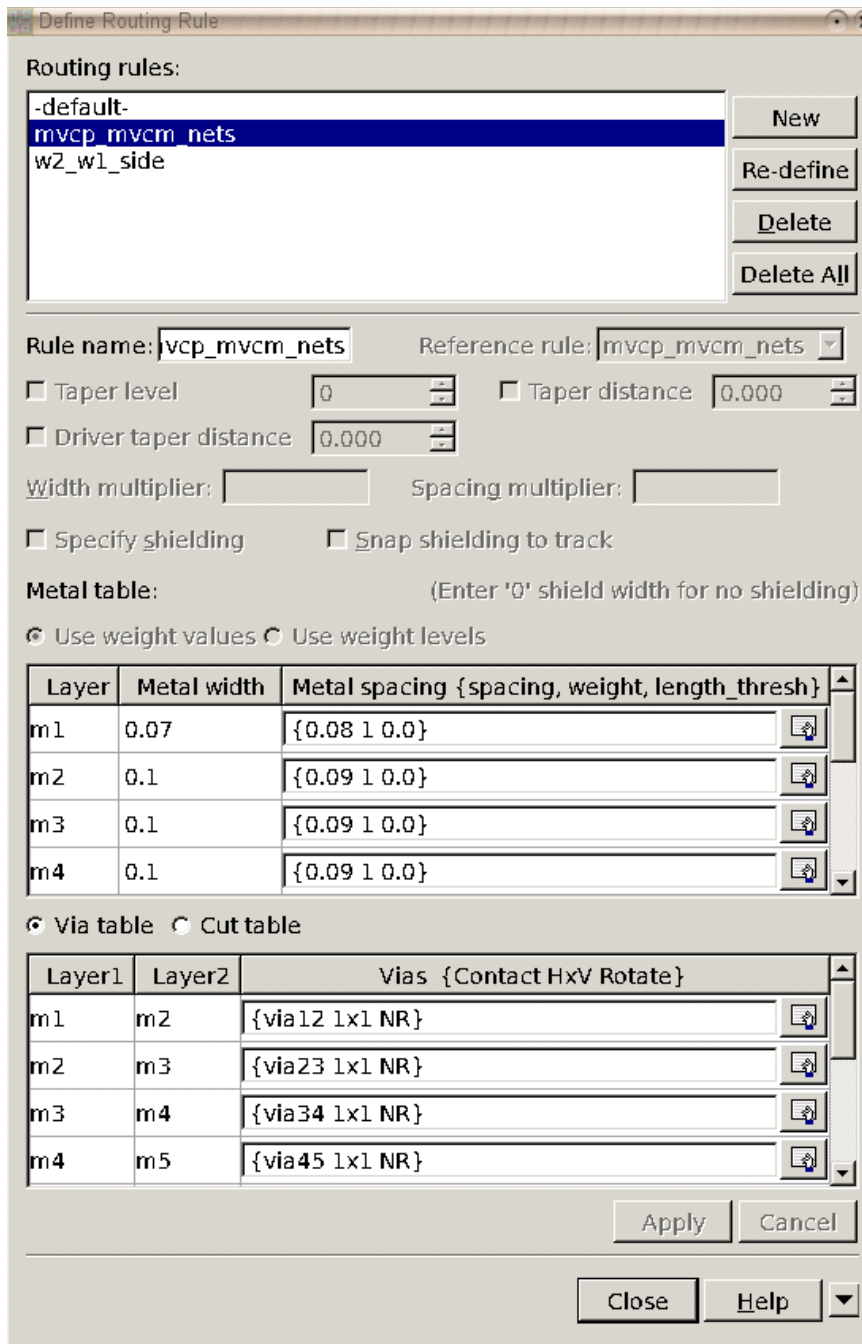


Figure 2 Define Routing Rule Dialog Box

- c. Click on the **mvcp\_mvcm\_nets** rule.
- d. Observe that layers m2-m5 have non-default rules for width set to 0.1u as shown in [Figure 3](#).



The dialog box is titled "Define Routing Rule". It contains a list of routing rules on the left, with "mvcp\_mvcm\_nets" selected. To the right of the list are buttons for "New", "Re-define", "Delete", and "Delete All".

Below the list, the "Rule name" is "vcp\_mvcm\_nets" and the "Reference rule" is "mvcp\_mvcm\_nets". There are input fields for "Taper level" (0), "Taper distance" (0.000), and "Driver taper distance" (0.000). There are also input fields for "Width multiplier" and "Spacing multiplier".

There are checkboxes for "Specify shielding" and "Snap shielding to track". Below these is a "Metal table" section with a note "(Enter '0' shield width for no shielding)". There are two radio buttons: "Use weight values" (selected) and "Use weight levels".

Layer	Metal width	Metal spacing {spacing, weight, length_thresh}
m1	0.07	{0.08 1 0.0}
m2	0.1	{0.09 1 0.0}
m3	0.1	{0.09 1 0.0}
m4	0.1	{0.09 1 0.0}

Below the metal table are two radio buttons: "Via table" (selected) and "Cut table".

Layer1	Layer2	Vias {Contact HxV Rotate}
m1	m2	{via12 1x1 NR}
m2	m3	{via23 1x1 NR}
m3	m4	{via34 1x1 NR}
m4	m5	{via45 1x1 NR}

At the bottom right are buttons for "Apply", "Cancel", "Close", and "Help".

Figure 3 Define Routing Rule Dialog Box

- e. Close the **Define Routing Rule** window.

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 2: Add and Open IC Compiler Libraries for this Design in Custom Designer

6. Type the following IC Compiler commands to find the nets associated with the above routing rules:

```
icc_shell> get_nets -filter "var_route_rule == w2_w1_side"  
{vreg}
```

```
icc_shell> get_nets -filter "var_route_rule == mvcp_mvcm_nets"  
{mvcm mvcp}
```

7. Close the IC Compiler design.

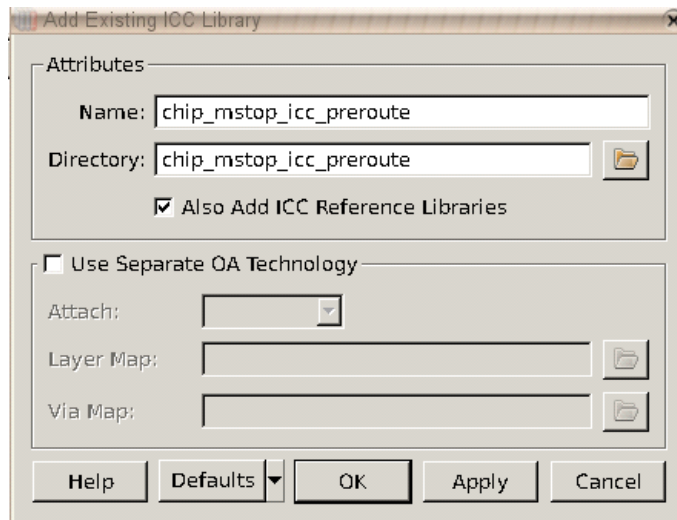
---

## Task 2: Add and Open IC Compiler Libraries for this Design in Custom Designer

In this task, you will add the IC Compiler libraries for this design to the Custom Designer platform to make them available for capturing design changes.

1. Make sure your current working directory is `<path>/CDICC_NonDefaultRules_Lab` and choose Custom Designer (CD).  
`unix% cdesigner &`
2. Choose the **Library Manager** tool.
3. In the **Library Manager**, choose **File > Add ICC Library**.  
This opens the **Add Existing Library** dialog box with default values shown in [Figure 4](#).
4. Using the browse button, choose the **chip\_mstop\_icc\_preroute** library, which adds the name to the **Directory** field.
5. Specify the **chip\_mstop\_icc\_preroute** in the **Name** field.

**Chapter 8: Tutorial: Constraint Rules Mapping**  
Task 2: Add and Open IC Compiler Libraries for this Design in Custom Designer



*Figure 4 Add Existing ICC Library Dialog Box*

6. Click **OK**.

The system adds this library and lists it in the `libs.defs` file.

7. The **Library Manager** now lists the IC Compiler library `chip_mstop_icc_preroute` and its reference libraries available to capture changes, as shown in [Figure 5](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 3: Open the Design Layout

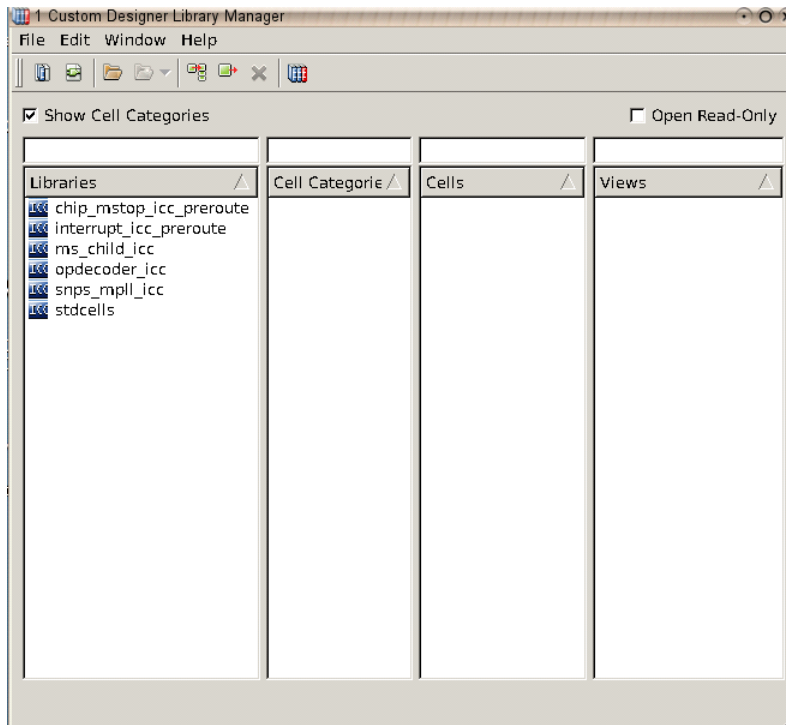


Figure 5 Custom Designer Library Manager Window

---

## Task 3: Open the Design Layout

In this task you will open a design originating in the IC Compiler tool and analyze the layout design editor configuration when opened directly.

1. In the **Library Manager** window, click on the **chip\_mstop\_icc\_preroute/cpu\_preRoute/layout**.
2. Double-click on the layout view to open the IC Compiler design, which opens the design in a layout window.

## Task 4: Review NDRs with the Custom Designer Layout Editor

In this task, you will learn how non-default rules (NDRs) created with the IC Compiler tool are mapped for use on the Custom Designer platform.

1. Open the **Constraint Editor** assistant.
  - a. Click the **Layers** constraint associated with Group **w2\_w1\_side**. This is equivalent to an IC Compiler non-default rule.

**Note:** For each IC Compiler non-default rule that is applied to any net, the Custom Designer platform creates a net group with the same name and adds the nets in compliance with the non-default rule to the group.
  - b. Observe that width values match IC Compiler values as shown in [Figure 6](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 4: Review NDRs with the Custom Designer Layout Editor

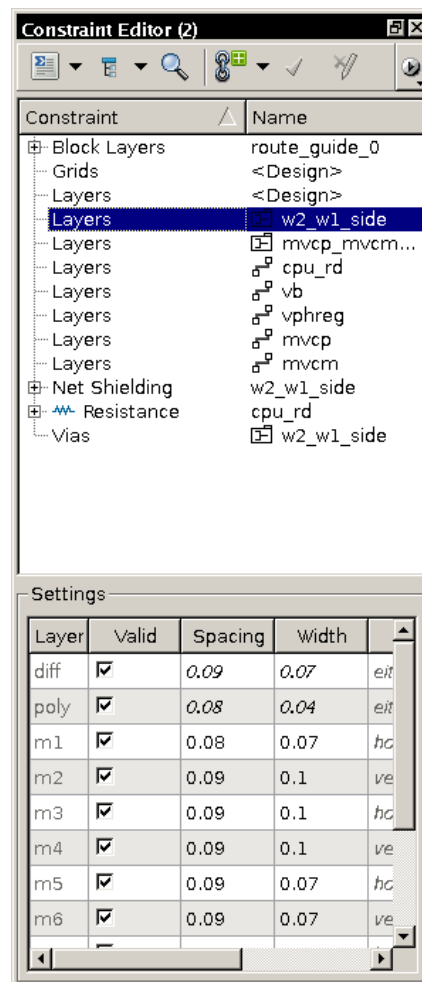


Figure 6 Constraint Editor Window

The shield spacing/width values associated with the IC Compiler NDR are also mapped to a shielding constraint.

**Note:** If the IC Compiler design has a single net/group of nets belonging to an NDR that defines shield width and spacing, then the NDR will be mapped to a net group with the same name and a shield constraint in Custom Designer with a shield net name. The shield net name will be set to the default ground net name.

- c. Click and expand the **Net Shielding** constraint associated with Group **w2\_w1\_side**.



- Observe that **Shield Net** is set to VSS, which is the ground net.
- Observe that shield width and spacing values match IC Compiler values as shown in [Figure 7](#).

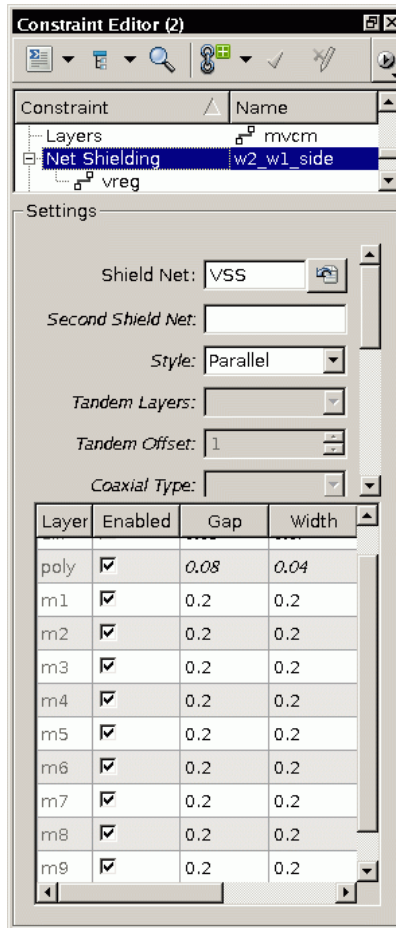


Figure 7 Constraint Editor Window

2. Similarly review the **Layers** constraint associated with the **mvcp\_mvcm\_nets** non-default rule. Observe that width values match IC Compiler values as shown in [Figure 8](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 5: Edit and Create NDRs and Shield Constraints

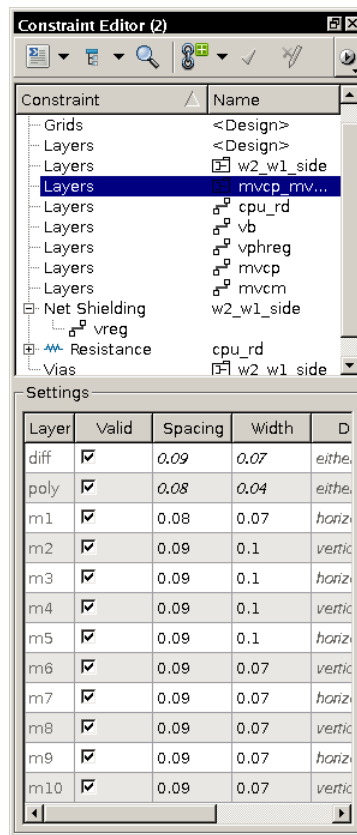




Figure 8 Constraint Editor Window

## Task 5: Edit and Create NDRs and Shield Constraints

In this task, you will edit the design by creating new non-default rules (NDRs) and constraints using Custom Designer's unique features.

Beginning with Custom Designer version I-2013.12-SP2, for IC Compiler designs:

- Shield constraint creation is supported for net groups. The shield constraint name option will not be available.
    - If the net group originated from IC Compiler through a routing rule, then when a new shielding constraint is created on that net group in Custom Designer, the shielding constraint is added to the routing rule when saving the design.
    - If the net group is created in Custom Designer, then on design save a new IC Compiler NDR is created with same name as the net group, with the signal and shield NDR values.
  - Shield constraint creation is restricted for individual nets.
1. To create a shielding constraint for net **mvcm**, the shielding constraint has to be created on the net group by following these steps:
    - a. Open the **Design Navigator** assistant.
    - b. Choose the **Nets** drop-down menu using the drop-down arrow .
    - c. Choose **Net by Group**.
    - d. Select the net group **mvcp\_mvcm\_nets**.
    - e. Use the **Create Constraint** button menu  to choose the **Net Shielding** constraint to apply to this net group, which creates a net shielding constraint (shown in [Figure 9](#)).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 5: Edit and Create NDRs and Shield Constraints

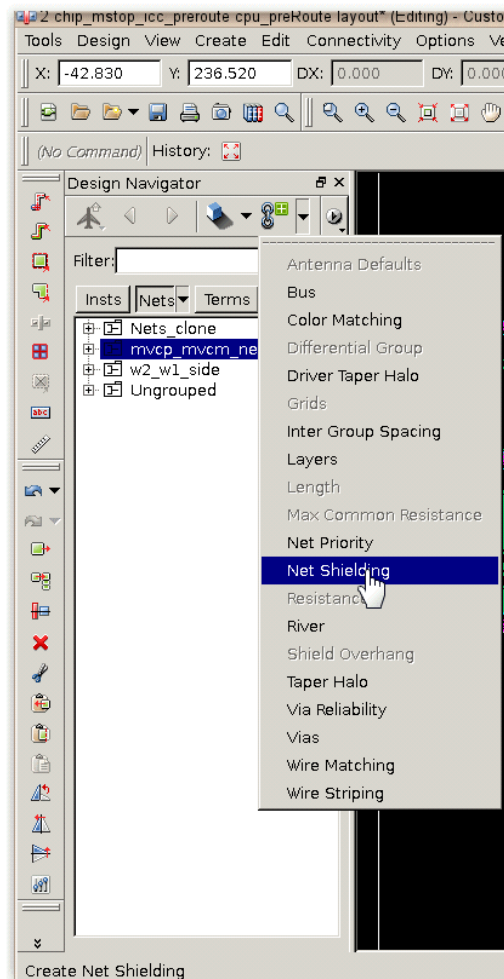


Figure 9 Creating Net Shielding Constraint in the Design Navigator

2. Modify the shielding constraint by following these steps:
  - a. Open the Constraint Editor assistant.
  - b. Select the **Net Shielding** constraint associated with the group **mvcp\_mvcm\_nets**.
  - c. Edit the **Shield Net**, **Default Width**, and **Default Gap** options as shown in [Figure 10](#).

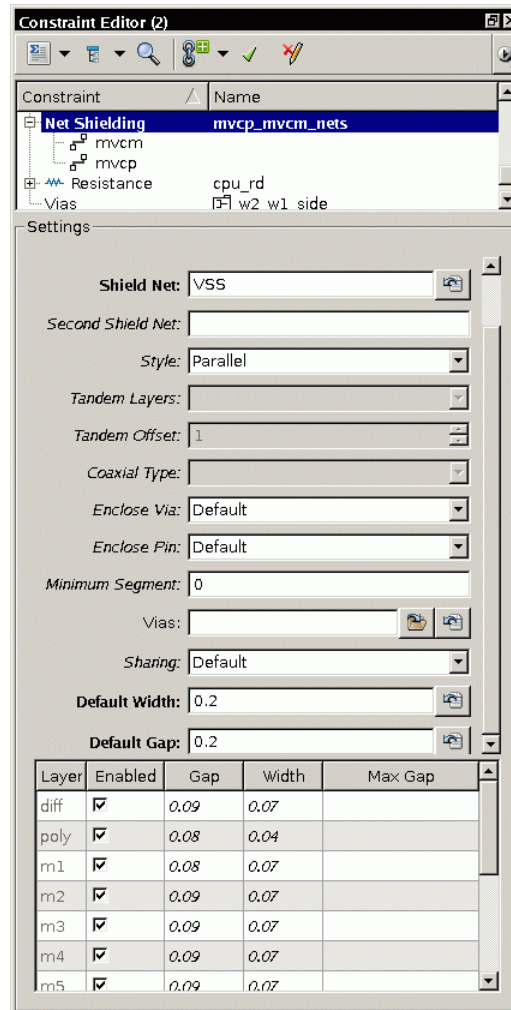


Figure 10 Editing Shield Constraints in the Constraint Editor Window


- d. Click **Apply** to save the changes.
3. In the following steps you will create an NDR and shield constraints for nets clk0p and clk0m. Because shield creation is restricted for nets, you will create a net group called clk\_diff and add nets clk0m and clk0p to this net group.

For nets clk0p and clk0m, create a net group and assign nets to it by following these steps:

- a. Open the **Design Navigator** assistant.

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 5: Edit and Create NDRs and Shield Constraints

- b. Choose the **Nets** drop-down menu using the drop-down arrow .
- c. Choose **All Nets**.
- d. In the **Nets** list, select the nets **clk0p** and **clk0m** by name.
- e. In the Design Navigator toolbar, click **Add to Group**.
- f. In the **Add to Group** dialog box, click **New**.
- g. In the text box, enter the name **clk\_diff** for the group.
- h. Select the group **clk\_diff**.
- i. Click **Add to Group** (shown in [Figure 11](#)).

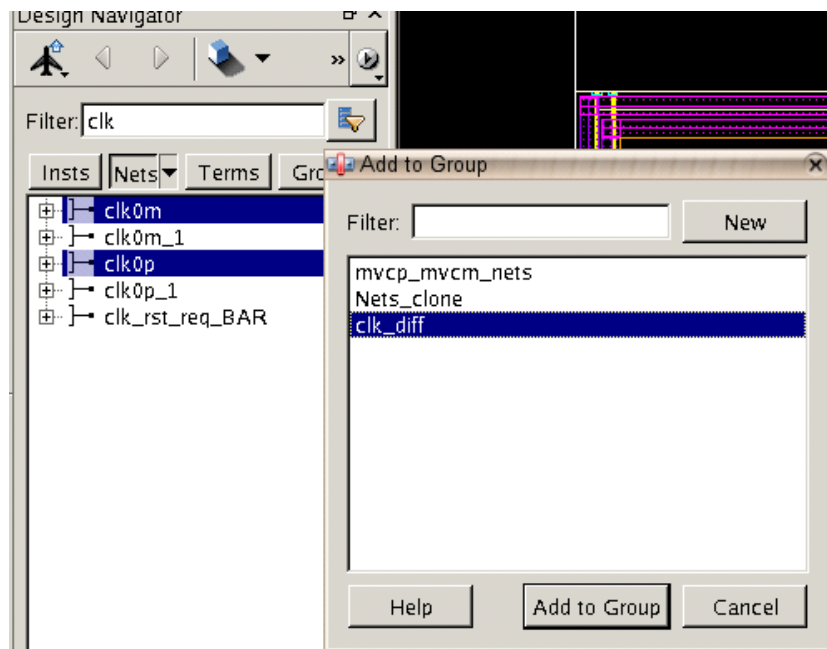



Figure 11 Creating a Net Group in the Design Navigator

- j. Close the dialog box.
4. For the net group **clk\_diff** created in previous steps, create an NDR with the following steps.

In the Design Navigator assistant:

- a. Choose the **Nets** drop-down menu using the drop-down arrow .

- b. Choose **Net by Group**.
  - c. Select the **clk\_diff** net group.
5. Open the Constraint Editor assistant.
  - a. Observe that there is no NDR defined on the net and thus the Constraint Editor assistant does not show any constraints.
  - b. Click the **Constraints** button  to open the list of available constraints (shown in Figure 12).
  - c. Select **Routing** from the list, which opens the list of available routing constraints (shown in Figure 12).
  - d. Choose **Layers** from the list, which adds the Layer constraint (shown in Figure 12).

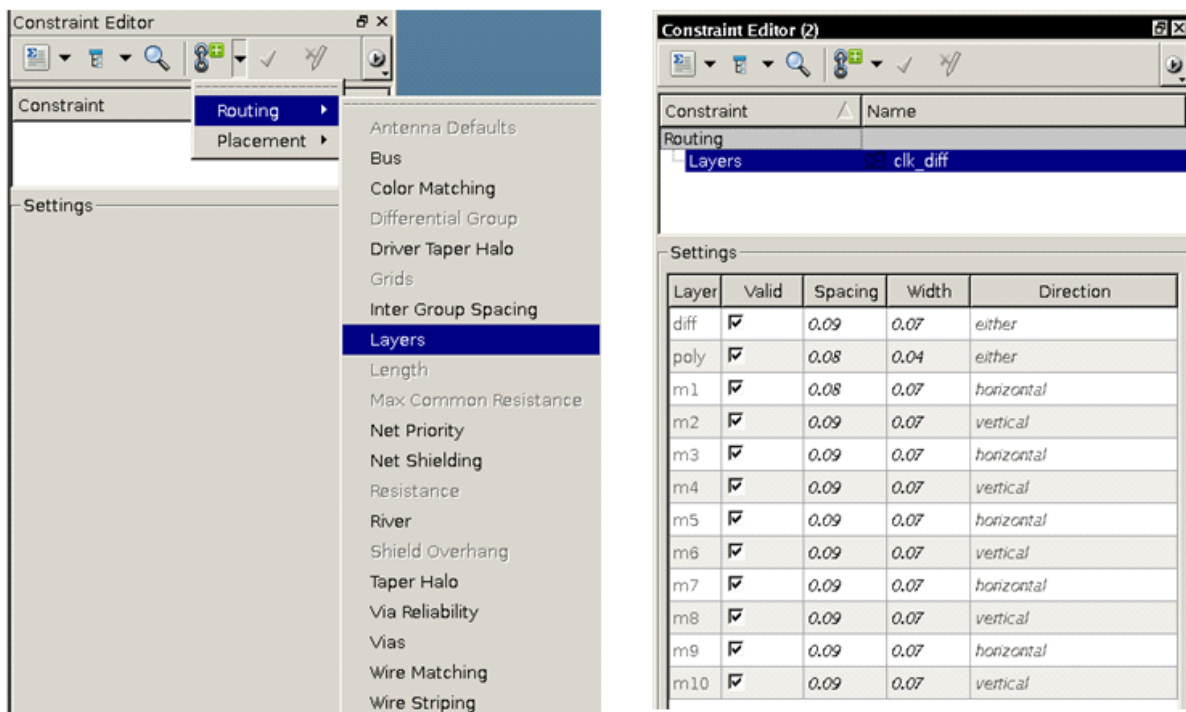


Figure 12 Creating an NDR in the Constraint Editor

6. In the Constraint Editor, modify the Layer constraints as follows:
  - a. Set the **Spacing** and **Width** of layers m2, m3, and m4 to 0.1u, as shown in Figure 13.

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 5: Edit and Create NDRs and Shield Constraints

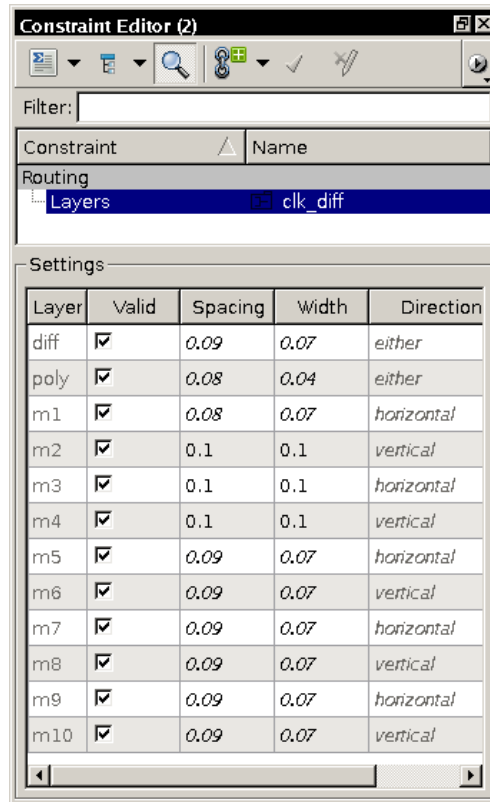


Figure 13 Modifying Layer Constraints in the Constraint Editor Window

- b. Click **Apply** to save the changes.
7. For the net group **clk\_diff** created in previous steps, create a shielding constraint with the following steps.
  - a. In the Design Navigator assistant, select the **clk\_diff** net group and create a shielding constraint.
  - b. In the Constraint Editor assistant, select the shielding constraint associated with **clk\_diff**.
  - c. Edit **Shield Net**, **Default Width**, and **Default Gap** options as shown in [Figure 14](#).



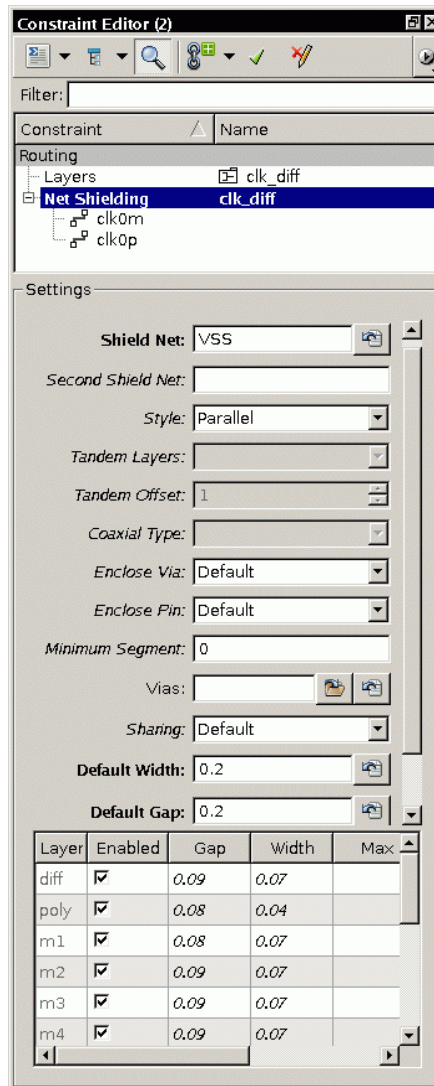


Figure 14 Modifying Shield Constraints in the Constraint Editor Window

- d. Click **Apply** to save the changes.
8. For the net **vphreg**, change the default width for metal layers m2 through m5 from 0.07 to 0.2.
9. Choose **Design > Save** to save the changes made to the design.
10. Choose **Design > Close** to close the design.

---

## Task 6: Evaluate Edits with the IC Compiler Tool

In this task you will evaluate the edits you have performed in the Custom Designer Layout Editor. You will check to make sure they are synchronized by using the IC Compiler tool.

1. Launch the IC Compiler tool using the following UNIX command.

**Unix%**`icc_shell -gui &`

2. From **File > Open Design:**
  - a. Select **chip\_mstop\_icc\_preroute** for **Library Name**.
  - b. Select **cpu\_preRoute** for **CEL**.

You will review the routing rules for net **mvcm**.

3. Use the IC Compiler commands below to find the nets associated with the routing rule mvcp\_mvcm\_nets:

**get\_nets -filter "var\_route\_rule == mvcp\_mvcm\_nets "**  
**{mvcp mvcm}**

Net **mvcm** remains part of the routing rule because a new shielding constraint was created for the net group in Custom Designer.

4. Choose **Define Routing Rule (Route > Routing Setup > Define Routing Rule)**.
  - a. Click on the **mvcp\_mvcm\_nets** rule.
  - b. Observe that metal width, spacing are same as original route rule and shield width, spacing 0.2u defined for all layers (as defined in [Task 5: Edit and Create NDRs and Shield Constraints](#)). See [Figure 15](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 6: Evaluate Edits with the IC Compiler Tool

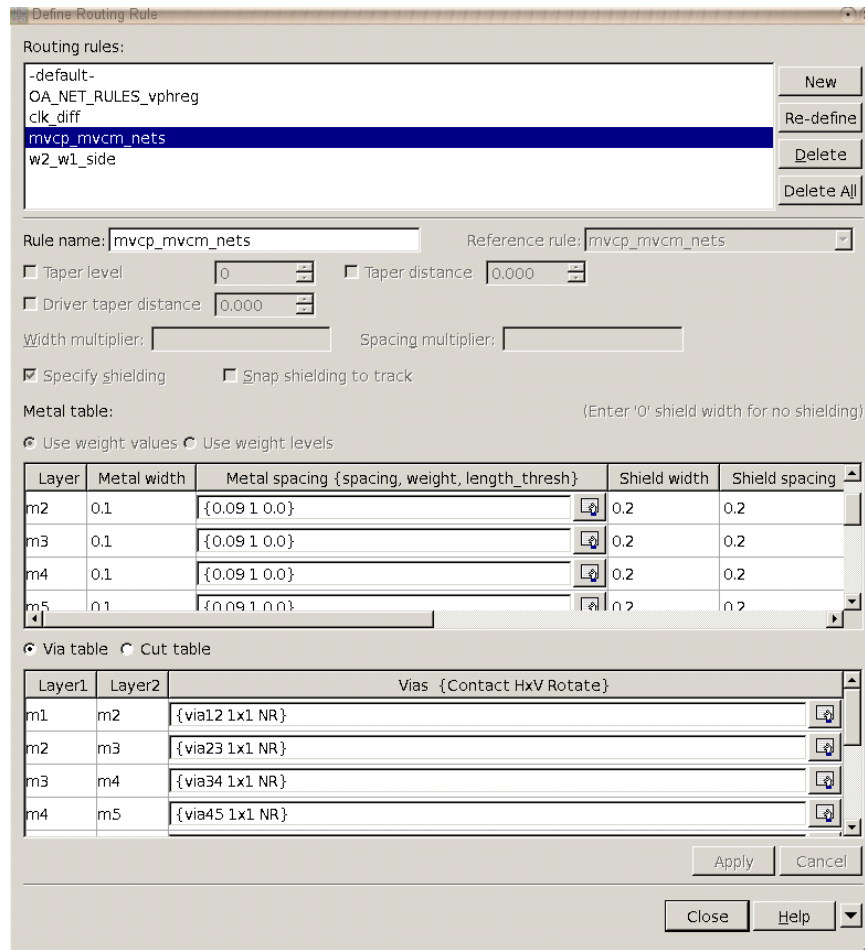


Figure 15 Define Routing Rule Window

- Observe that there is a new routing rule called **OA\_NET\_RULES\_vphreg**. When signal width/spacing is modified for a net, on design save the net will be assigned to a net-specific route rule (that is, the route rule name begins with "OA\_NET\_RULES\_").
- Click on the **OA\_NET\_RULES\_vphreg** rule and review the values. See [Figure 16](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 6: Evaluate Edits with the IC Compiler Tool

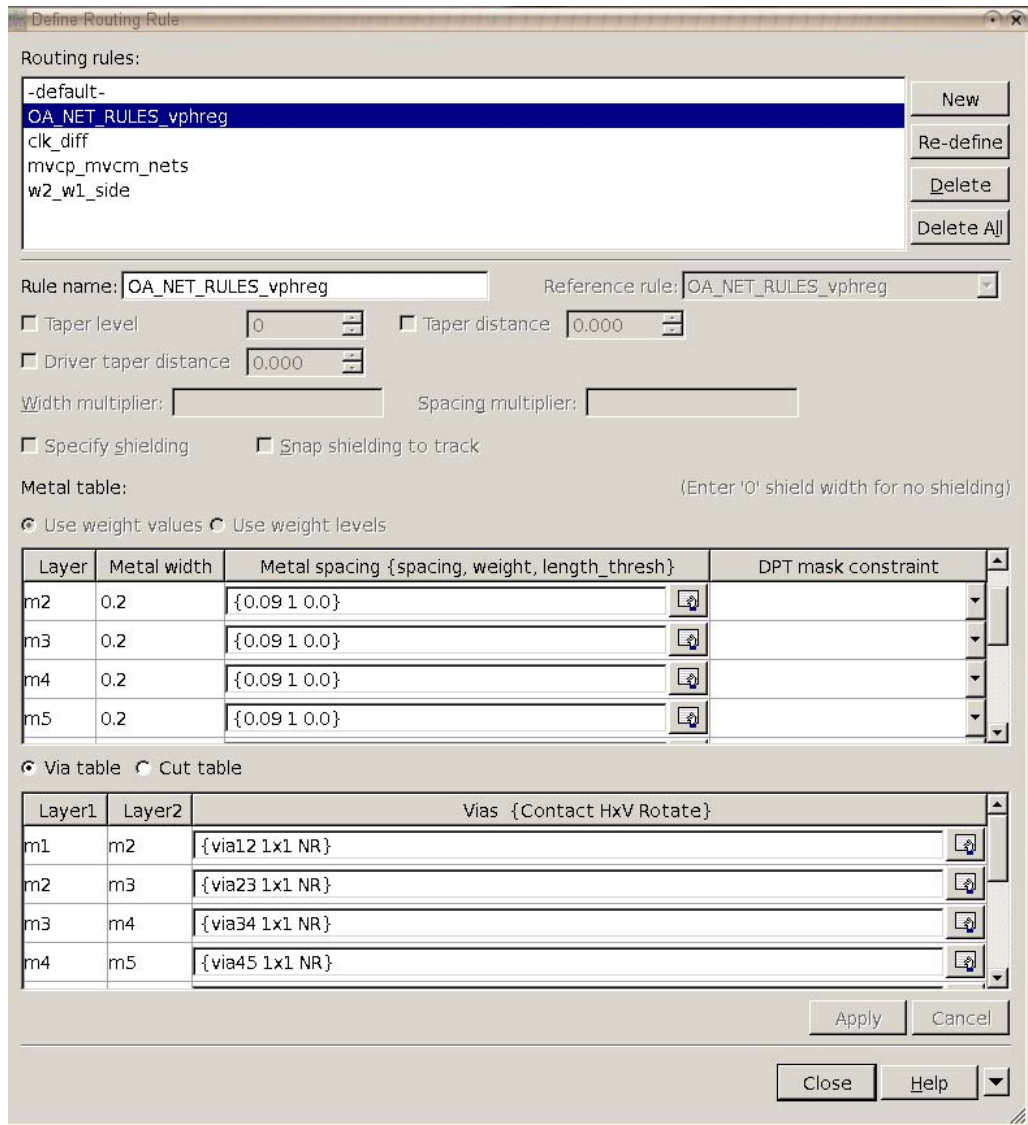


Figure 16 Define Routing Rule Window

7. Observe that there is also a routing rule named **clk\_diff**.
8. Click on the **clk\_diff** rule.

The **clk\_diff** shielding group created in Task 5 is saved to IC Compiler as a routing rule with appropriate constraints, where the net routing rule contains all the nets assigned to the routing rule. See [Figure 17](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 6: Evaluate Edits with the IC Compiler Tool

Define Routing Rule

Routing rules:

- default-
- OA\_NET\_RULES\_mvcm
- clk\_diff**
- mvcp\_mvcm\_nets
- w2\_w1\_side

Rule name:  Reference rule:

☐ Taper level  ☐ Taper distance

☐ Driver taper distance

Width multiplier:  Spacing multiplier:

☒ Specify shielding ☐ Snap shielding to track

Metal table: (Enter '0' shield width for no shielding)

☒ Use weight values ☐ Use weight levels

Layer	Metal width	Metal spacing {spacing, weight, length_thresh}	Shield width	Shield sp
m1	0.07	{0.08 1 0.0}	0.2	0.2
m2	0.1	{0.10 1 0.0}	0.2	0.2
m3	0.1	{0.10 1 0.0}	0.2	0.2
m4	0.1	{0.10 1 0.0}	0.2	0.2
m5	0.07	{0.09 1 0.0}	0.2	0.2
m6	0.07	{0.09 1 0.0}	0.2	0.2

☒ Via table ☐ Cut table

Layer1	Layer2	Vias {Contact HxV Rotate}
m1	m2	{via12 1x1 NR}
m2	m3	{via23 1x1 NR}
m3	m4	{via34 1x1 NR}
m4	m5	{via45 1x1 NR}
m5	m6	{via56 1x1 NR}
m6	m7	{via67 1x1 NR}
m7	m8	{via78 1x1 NR}

Figure 17 Define Routing Rule Window

9. Close the **Define Routing Rule** window.

10. Use the following IC Compiler command to validate the nets associated with this rule:

```
icc_shell> get_nets -filter "var_route_rule == clk_diff"  
{clk0p clk0m}
```

---

## Task 7: Edit Routing Rules in IC Compiler

In this task you will learn to edit the routing rules created in Custom Designer and save it back to IC Compiler.

1. Choose **Define Routing Rule** (**Route > Routing Setup > Define Routing Rule**).
2. Click on the **OA\_NET\_RULES\_vphreg** rule
3. Click **Re-define** to modify the rule
4. Change the **Shield width** and **Shield spacing** values for layers m2-m4 to 0.1u as shown in [Figure 18](#).

## Chapter 8: Tutorial: Constraint Rules Mapping

### Task 7: Edit Routing Rules in IC Compiler

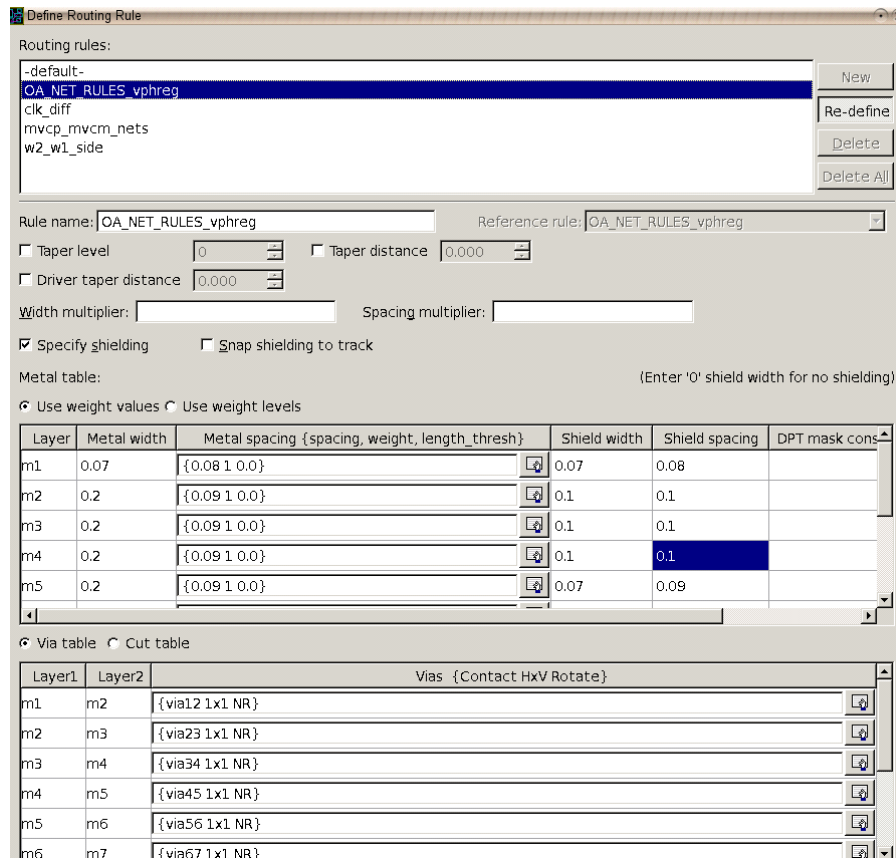


Figure 18 Define Routing Rule Window

- Click **Apply** to save the changes.
- Use the following IC Compiler command to validate the nets associated with this rule. You will receive the warning shown.

```
icc_shell> get_nets -filter "var_route_rule == OA_NET_RULES_vphreg"
```

```
"Warning: No net objects matched '*' (SEL-004)"
```

In IC Compiler, editing a routing rule basically redefines the routing rule and doing so causes the removal of the nets associated with the routing rule. Thus, after redefining the routing rule, the nets have to be reassigned the routing rule.

```
icc_shell> set_net_routing_rule vphreg -rule OA_NET_RULES_vphreg
```

- Save and close the design.

8. Optionally, evaluate the edits in Custom Designer.
9. Exit IC Compiler.

---

## Summary

- IC Compiler NDRs (Non-Default Rules) are mapped to equivalent OA as a net group with appropriate constraints (non-default width and spacing, shield width and spacing) and the net group contains all the nets assigned to the route rule.
- When the IC Compiler design has a single ground net and if the IC Compiler design has an NDR that defines shield width and spacing, then the NDR will be mapped to a shield constraint in Custom Designer with shield net name set to ground net name.
- When the IC Compiler design has multiple ground nets and if the IC Compiler design has an NDR that defines shield width and spacing, then the NDR will be mapped to shield constraint in Custom Designer without a shield net name and you have to specify the desired shield net name for the routing purposes.
- When the IC Compiler design has a group of nets belonging to an NDR and for one of the nets (e.g., netA) a signal width/spacing is modified in Custom Designer, then netA is assigned to a net-specific routing rule (that is, the routing rule name begins with “OA\_NET\_RULES\_”) and is not part of the original routing rule and net group. This happens during design save.
- For IC Compiler designs, shielding constraints can be created only for net groups. Shielding constraint creation is disabled for nets.

---

## Limitations

- If a shield or NDR constraint is created for a net in Custom Designer and if the design was saved to IC Compiler, then deleting the constraint does not delete the NDR rule in IC Compiler.
  - Use the IC Compiler `remove_routing_rules` command to remove the complete rule.



- If the rule has to be partially removed, use the **Re-define** option in the **Define Routing Rule** window to update the rules.

## **Chapter 8: Tutorial: Constraint Rules Mapping**

### Limitations

## Setting the User Environment

---

*Custom Designer - IC Compiler Co-Design supports various flows. This appendix describes how to set up the Custom Designer and IC Compiler environments for different scenarios of technology information availability.*

This chapter covers:

- [Tech Mapping between the Custom Designer Platform and the IC Compiler Tool](#)
- [Export to the IC Compiler Tool](#)
- [Import from the IC Compiler Tool](#)
- [Pre-Route Flow](#)
- [Setup Using LEF Files](#)

---

### Tech Mapping between the Custom Designer Platform and the IC Compiler Tool

This section provides information about the basic mapping between the Custom Designer platform and IC Compiler technology layers and their display attributes.

---

#### Custom Designer Layer Purpose Pair (LPP) to IC Compiler Layer: DataType Mapping

The IC Compiler tool's physical data are organized on layers and are assigned a *dataType*. On the Custom Designer platform, the corresponding terms are *layer* and *purpose*.

## Appendix A: Setting the User Environment

### Tech Mapping between the Custom Designer Platform and the IC Compiler Tool

The IC Compiler Layer specifies a layer with a data type number of 0. The Layer corresponds to the Custom Designer Layer Purpose Pair (LPP) of the matching Custom Designer layer with the Custom Designer drawing purpose, which is a system-reserved purpose defined in the `techLayers` section of the Custom Designer technology file.

**Note:** Milkyway schema 7 and schema 8 support `extend_mw_layers`, which extends Milkyway database layer number support to 4095 layers, using layers 4001 through 4095 as the system-reserved layers. See the IC Compiler documentation for more details.

*Table 3 IC Compiler Tool -Custom Designer Layer Mapping*

IC Compiler Tech Rule Attributes	Custom Designer Tech Constraint
<pre>Layer &lt;layerName&gt; {   layerNumber = &lt;value&gt; } LayerDataType &lt;dataTypeName&gt;{   layerNumber = &lt;value&gt;   dataTypeNumber= &lt;dtvalue&gt; }</pre>	<pre>techLayers(   ( &lt;layerName&gt;      &lt;value&gt; ) ) techPurposes(   ( drawing -1      drw      )   ( dataTypeName dtvalue      ) )</pre> <p>NOTE: drawing is a reserved purpose name</p>

When mapping from IC Compiler to Custom Designer, all the IC Compiler `via<m> maskName` mask names map to "cut" and all the `metal<n> maskName`

mask names become "metal". The IC Compiler polyCont mask name also maps to "cut".

*Table 4 IC Compiler Tool -Custom Designer Layer Mapping*

IC Compiler Tech Rule Attributes	Custom Designer Tech Constraint
<pre> Layer &lt;layerName&gt; {   layerNumber = &lt;value&gt;   maskName = &lt;value&gt; } </pre>	<pre> layerRules(   functions(     ;( layer function [maskNumber])     &lt;layerName&gt;    &lt;value&gt;    30 )   ) ) </pre>

The following table maps IC Compiler tool mask names to Custom Designer platform material names.

*Table 5 IC Compiler Tool -Custom Designer maskName Mapping*

IC Compiler Mask Name	Custom Designer Material Name
poly	poly
via1...vian	cut
metal1.. metaln	metal
nwell	nWell
pwell	pWell
diffusion	diffusion
polyCont	cut

## Custom Designer Platform to IC Compiler Display Attribute Mapping

Custom Designer Display Resource data, colors, lineStyles, stipples, packets and packetAliases, are all defined in the context of a "display" to display a design, such as a graphics display, laser printer or pen plotter.

## Appendix A: Setting the User Environment

### Tech Mapping between the Custom Designer Platform and the IC Compiler Tool

IC Compiler display resource data has no context; only one set of display resource definitions exists in an IC Compiler library. The IC Compiler tool also lacks the concepts of "packet" and "packetAlias."

IC Compiler tool to Custom Designer platform translation maps the Milkyway display resource data to the "display" of the corresponding Custom Designer `display.drf` file.

Through packet and packetAlias definitions, the Custom Designer display resource data defines the drawing attributes of the LPPs.

*Table 6 IC Compiler Tool -Custom Display Attributes Mapping*

IC Compiler Tech Rule Attributes	Custom Designer Tech Constraint
<pre> Stipple"solid" { width= 16 height= 16 ..... }  Color15 { name= "cyan" rgbDefined = 0 }  Stipple"dot4" { width= 16 height= 16 }  Layer "m1" { layerNumber = 8 visible= 1 selectable = 1 blink = 0 color = "cyan" lineStyle= "solid" pattern= "dot4"  ..... } </pre>	<pre> In display.tcl: dr::createLineStyle solid -display \$display_object -width 1 -pattern [list 1 1 1 ]  dr::createColor cyan -display \$display_object -red 0 -green 255 -blue 255  dr::createStipple dot4 -display \$display_object -pattern {.....}  dr::createPacket m1 -display \$display_object -stipple dot4 \         -lineStyle solid -fill cyan -outline cyan  In Custom Designer technology file:  techDisplays(     ;( LayerName Purpose Packet Vis Sel Con2ChgLy DrgEnbl Valid )  ;( ----- ) ----- ) ( m1    drawing    m1    t t nil t t ) ) ) </pre>

---

## Export to the IC Compiler Tool

This section covers the set-up required while exporting a Custom Designer database to create a new IC Compiler database or write to an existing IC Compiler database.

---

### Scenario 1: Exporting without Available IC Compiler Technology or Design Data

In this scenario, there is a Custom Designer design library. There is no IC Compiler related data available; that is, there is no IC Compiler technology file or IC Compiler library.

For the given Custom Designer library, the user requirement could be either of the following:

- Generate an IC Compiler technology file [OR]
- Generate an IC Compiler technology file and IC Compiler library.

These options are described in the following sections.

#### Generate an IC Compiler Technology File

The `cd2icc_tech` executable can be used for this purpose.

This can be run outside of the Custom Designer platform from a UNIX terminal, which allows you to quickly create an IC Compiler technology file.

The following is an example that exports an IC Compiler tech file from a Custom Designer library.

```
UNIX%cd2icc_tech -oaLib snps_mpll -techFile mw.tff
```

**Note:** The Custom Designer library must have the tech database or a tech library reference in order to generate an IC Compiler technology file.

#### Generate IC Compiler Tech File and Library

To create an IC Compiler technology file and a library, the Export To ICC feature must be used.

## Appendix A: Setting the User Environment

### Export to the IC Compiler Tool

This feature creates an IC Compiler technology file and also an IC Compiler library for the given Custom Designer library.

You can choose **File > Export > To ICC** from the Console window, which allows you to export the Custom Designer library to an IC Compiler library.

Figure 19 shows a Custom Designer library specified under the **Input** section of the **Main** tab. The **OA Library** you specify here is exported to an IC Compiler library, which is specified in the **Output** section under **ICC Library**.

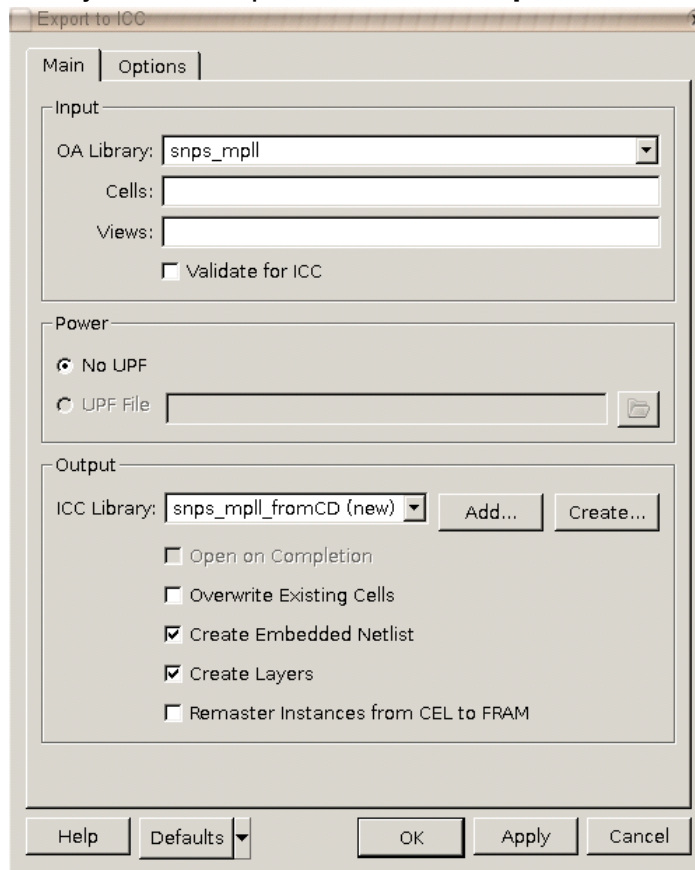


Figure 19 Export To ICC Dialog Box

During the export process, the tool creates the `./iccbridge` directory, which contains all the output files that are generated.

The exported IC Compiler technology file can be identified by the name `./iccbridge/cd2icc_tech.tff`.

The IC Compiler library is created and placed under the CWD directory.



---

## Scenario 2: Exporting to an Existing IC Compiler Library with Matching Layers

In this scenario, there is a Custom Designer design and a technology database exists for it. There is an IC Compiler library available to export the design from Custom Designer into the existing IC Compiler library.

The tech database for both the Custom Designer platform and the IC Compiler tool matches in all respects as described below:

- The layer names/numbers are same for both the IC Compiler and Custom Designer versions.
- For the Custom Designer purposes associated with each layer, there is an equivalent IC Compiler dataType of the same name associated with the layer.

During the Export process, the tool checks to see if the Custom Designer layer names in the tech database match with the IC Compiler layer names.

If the layer names are matched, then no layer is created and they are automatically mapped by name.

Similarly, for the purpose, if an IC Compiler dataType name already exists in the destination technology file, no creation is necessary.

There would then be no environment setup required for this case.

An example follows.

### Example 1

For the Custom Designer design library `snps_mpl1`, the LPP m1 drawing is on lpp number 8:-1.

There is already an IC Compiler library `snps_mpl1_icc`, which has the metal1 layer (identified by the mask name "metal1") with layer name m1 on layer number 8 and dataType 0.

Similarly, the other layers also match.

When the library is exported to IC Compiler, the tech layer match is identified by the layer names. A mapping file is generated during the process.

The layer mapping file is available in the `iccbridge` directory. The file is called `layer.map`.

## Appendix A: Setting the User Environment

### Export to the IC Compiler Tool

For the given example, the `layer.map` will look like this:

CDLayerName	CDLayerPurpose	ICCLayerNumber	ICCLayerDatatypeNumber
m1	drawing	8	0

---

### Scenario 3: Exporting to an Existing IC Compiler Library with Different Layer Names

In this scenario, there are existing Custom Designer design and tech databases. There is also an IC Compiler library available for export.

**Important:** In this case the layer names are mismatched between the Custom Designer and IC Compiler technology databases.

By default, during the export process, for the Custom Designer layers that do not have an IC Compiler layer with same name, a new layer is created in the IC Compiler technology database.

During the export process, the tool checks the Custom Designer layer names in the tech database against the IC Compiler layer names.

- If the layer number is not already in use in the destination IC Compiler technology file, a new layer is created in the destination IC Compiler technology file using the source layer name and number.
- If the layer number is in use, a new layer is created in the destination IC Compiler technology file with the source layer name and an unused layer number.

**Note:** IC Compiler tool layer names are limited to 31 characters. If an OA layer name is longer than this, then the name is truncated to 27 characters and '\_n' is appended where 'n' is 1 to 3 digits and is chosen, in order to generate a unique name.

To avoid creating new layers during the export process, the required setup would be to create a `layer.map` file to resolve the layer name mismatches.

#### Example 2

In this example, the layer names do not match between the Custom Designer and IC Compiler tech data.

In the Custom Designer library `snps_mpl1`, metal layers are on the LPPs: m#drawing, where #=1, 2... 10.

In the IC Compiler library `snps_mpll_icc`, metal layers are on the IC Compiler tech layers: M#: dataType 0, where #=1, 2,...,10 respectively.

Table 7

Custom Designer Layers	IC Compiler Layers
m1 (8) :drawing (-1)	M1 (8) : dataType 0
m2(10):drawing (-1)	M2(10): dataType 0
.....	
m10(85):drawing(-1)	M10(85):dataType 0

The design `snps_mpll/msip_dcc/layout` is exported to `snps_mpll_icc`, as shown in [Figure 20](#).

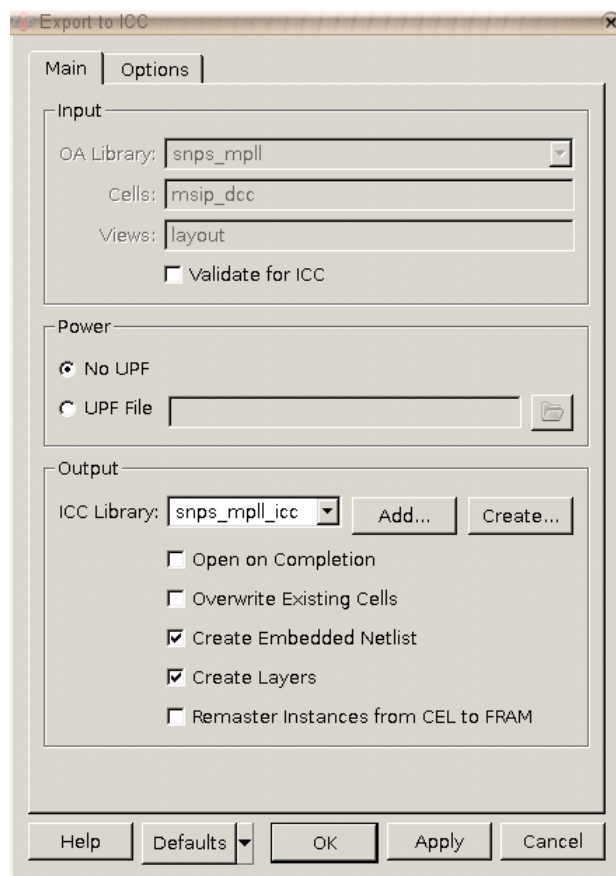


Figure 20 Export To ICC Dialog Box

## Appendix A: Setting the User Environment

### Export to the IC Compiler Tool

If no layer map file is provided during export, a `layer.map` file (similar to the one below) is generated by the tool.

CDLayerName	CDLayerPurpose	ICCLayerNumber	ICLayerDatatypeNumber
m1	drawing	52	0
m10	drawing	51	0
m2	drawing	50	0
m3	drawing	48	0
m4	drawing	47	0
m5	drawing	46	0
m6	drawing	45	0

For the Custom Designer LPPs m#: drawing, which do not have an IC Compiler layer with same name, a new layer is created in the IC Compiler technology database on the available unused layers.

For the given example, to make sure that the tool uses the existing IC Compiler layers, a layer mapping file must be used to achieve the correct layer mapping.

The `layer.map` file used to resolve the mismatch is shown below:

CDLayerName	CDLayerPurpose	ICCLayerNumber	ICLayerDatatypeNumber
m1	drawing	8	0
m2	drawing	10	0
m3	drawing	12	0
m4	drawing	14	0
m5	drawing	70	0
m6	drawing	72	0
m7	drawing	74	0
m8	drawing	76	0
m9	drawing	78	0
m10	drawing	85	0

The layer map file has to be specified in the **Export To ICC** dialog, in the **Options** tab as shown in [Figure 21](#):

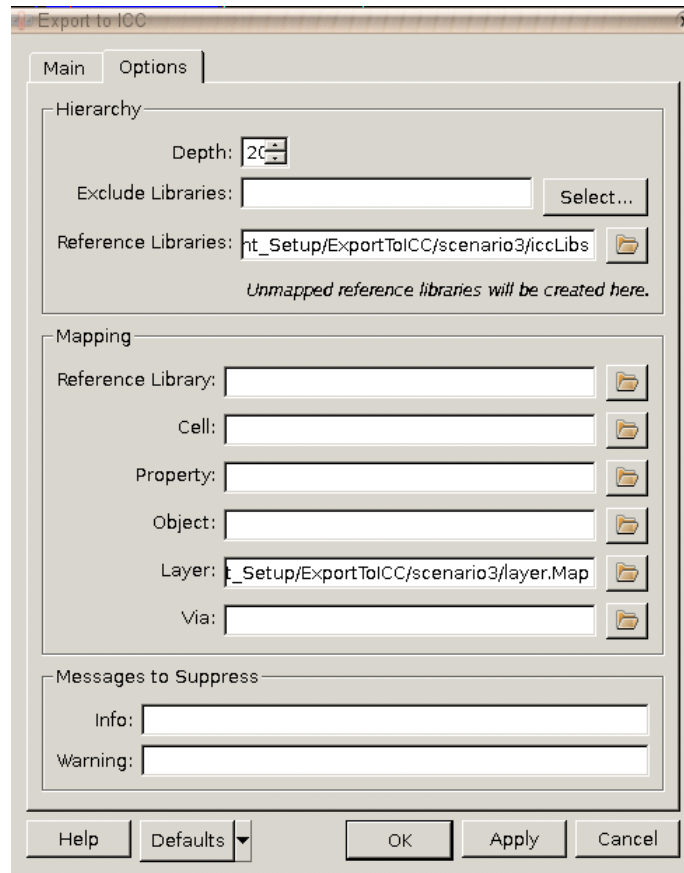


Figure 21 Export To ICC Dialog Box

During the export process, the tool uses the user-specified layer mappings to map the given Custom Designer data to IC Compiler data.

---

## Scenario 4: Exporting to an Existing IC Compiler Library with No Matching Layers

In this scenario, there is a Custom Designer design and a tech database. There is also an IC Compiler library available for export.

In this case, there are some Custom Designer LPPs that do not have an equivalent IC Compiler layer: data type match.

## Appendix A: Setting the User Environment

### Export to the IC Compiler Tool

There are three methods that allow for creation of the missing layers/purposes in the target IC Compiler technology:

- Use the automated behavior of **Export To IC Compiler**.
- Use the `layer.map` file.
- Create a new technology file and edit the existing IC Compiler technology file to include the missing layers and purposes.

### Using the Automated Method of Exporting to IC Compiler

During the export process, the tool checks the Custom Designer layer names in the tech database against the IC Compiler layer names:

- If the layer number is not already in use in the destination IC Compiler technology file, a new layer is created in the destination IC Compiler technology file with the source layer name and number.
- If the layer number is in use in the destination IC Compiler technology file, then a new layer is created in the destination IC Compiler technology file with the source layer name and an unused layer number.

The same process is also applied for layer purposes.

- If the purpose number is not already in use in the destination IC Compiler technology file, a new dataType is created in the destination IC Compiler technology file with the source purpose name and number. Otherwise, a new purpose is created in the destination IC Compiler techfile with the source purpose name and an unused dataType number.
- IC Compiler LayerDataType names are constructed from the Custom Designer purpose as '<purpose>\_cd###', where 'purpose' is the name of the Custom Designer purpose.

**Note:** Datatype names in MW are limited to 31 characters. If an OA purpose with more than 27 characters is found, then the name used in MW is truncated to 27 characters and '\_nnn' is appended, where 'nnn' is 1 to 3 decimal digits selected to be unique for all the datatypes created.

### Example 3

For the Custom Designer library `snps_mpll`, the tech and design database has

- shapes on the LPP "m# drawing," where #=1,2,3 and 4
- pin shapes on the LPP "m# pin".

The IC Compiler library `snps_mpll_icc` is available for export, and its technology database contains the equivalent metal layer definitions for LPPs: "m# drawing," but no `dataType` definitions for the purpose pin.

Table 8

Custom Designer Layers	IC Compiler Layers
m1 (8) :drawing (-1)	M1 (8) : dataType 0
m1(8):pin (251)	m1 (8):??
m2(10) :drawing(-1)	m2(10):dataType 0
m2(10):pin (251)	m2 (10):??

When the design `snps_mpll/msip_dcc/layout` is exported from the Custom Designer platform to the IC Compiler tool, the tech layer matching is resolved by the layer names.

The pin purpose is a reserved purpose in Custom Designer and has the purpose number as 251. So, in the IC Compiler tool, this is mapped to metal layer number: `dataType 251`.

The generated layer map file looks like the following:

m1	pin	8	251
m2	pin	10	251
m3	pin	12	251
m4	pin	14	251

## Appendix A: Setting the User Environment

### Export to the IC Compiler Tool

A new dataType 'pin\_cd###' is created for each metal layer, which has a pin purpose as shown :

```
Layer      "m1" {
    layerNumber    = 8
    maskName       = "metal1"
    .....
}
LayerDataType "pin_cd011" {
    layerNumber    = 8
    dataTypeNumber = 251
    .....
}
```

### Using the Layer Mapping File

To map the missing layers in the IC Compiler tool to a specific layer number and dataType, you can provide an explicit mapping using the `layer.map` file.

If the destination layer number/dataType number provided in the `layer.map` file does not exist, it is created.

#### Example 4

For the example described in Example 3, assume that you need to map the LPP "m# pin" to IC Compiler layer m#(#):dataTypeNumber(1).

Table 9

Custom Designer Layers	IC Compiler Layers
m1 (8) :drawing (-1)	m1 (8) : dataType 0
m1(8):pin (251)	m1(8): pin (1)

So, a `layer.map` file has to be created as shown:

m1	pin	8	1
m2	pin	10	1
m3	pin	12	1
m4	pin	14	1

When the export process is performed for the design `snps_mpll/msip_dcc/layout`, the layer map file has to be specified in the **Export To ICC** dialog



under the **Options** tab, as shown in Figure 22:

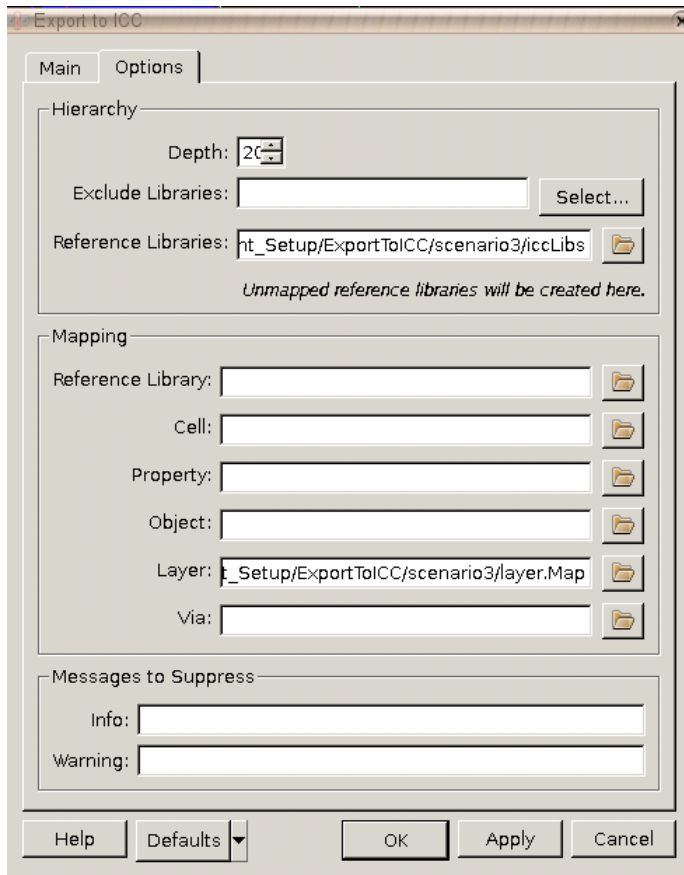


Figure 22 Export To ICC Dialog Box

The tool uses the specified layer mapping file and creates a new dataType 'pin\_cd###' on the specified numbers. Layer definitions are then created for each metal layer, which has the pin purpose, where 'pin' is the name of the Custom Designer purpose.

The dumped IC Compiler technology file looks like the following:

```
Layer      "m1" {
    layerNumber      = 8
    maskName         = "metal1"
    .....
}
LayerDataType "pin_cd011" {
    layerNumber      = 8
    dataTypeNumber   = 1
    .....
}
```

## Import from the IC Compiler Tool

This section covers the different scenarios for importing an IC Compiler technology database to create a new Custom Designer database or to update an existing one.

---

### Scenario 1: Importing without Available Custom Designer Technology or Design Data

In this scenario, there is an IC Compiler design library. There is no Custom Designer technology file or Custom Designer library.

For the given IC Compiler library, your requirement could be either one of the options that follow:

- Generate a Custom Designer technology file [OR]
- Generate a Custom Designer technology file and Custom Designer library.

#### Generate a Custom Designer Technology File

The `icc2cd_tech` executable can be used for this purpose.

You can run this outside the Custom Designer platform, in a UNIX window, which allows you to quickly create a Custom Designer technology file.

The following is an example that exports a Custom Designer technology file from an IC Compiler Design library:

```
UNIX: icc2cd_tech -mwLib opdecoder_icc -techFile cd.tf
```

**Note:** The IC Compiler library should have the technology information to generate the Custom Designer technology file.

#### Generate Custom Designer Technology File and Library

To create a Custom Designer technology file and a library, the **Import from ICC** feature has to be used.

This feature creates a Custom Designer technology file and a Custom Designer library for the given IC Compiler library.

You can choose **File > Import > From ICC** from the console window, which allows you to import the IC Compiler library to a Custom Designer library.

Figure 23 shows the IC Compiler library, which is specified under **Input section: ICC Library**. This IC Compiler library is imported to a Custom Designer library, which is specified in the **Output section under OA Library**.

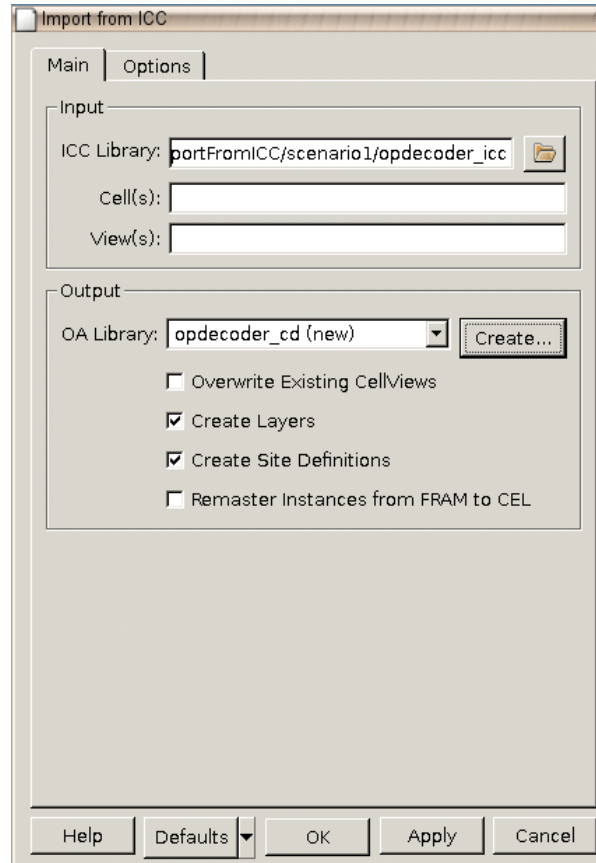


Figure 23 Import from ICC Dialog Box

During the import process, the tool creates the `iccbriidge` directory, which has all of the output files generated.

The Custom Designer technology file can be identified by the name `icc2cd_tech.tf`.

The Custom Designer library is created and placed under the `oaLibs` directory.

---

## **Scenario 2: Importing to an Existing Custom Designer Library with Matching Layers**

In this scenario, there is an IC Compiler design library database. There is also a Custom Designer library available for import.

**Important:** The Custom Designer tech database matches with IC Compiler technology in all respects.

During the import process, the tool checks the IC Compiler layer names against the Custom Designer tech layer names.

If the layer names match, no layers are created, as the layer mapping is done by name.

Similarly for the dataType, if a purpose name already exists in the destination Custom Designer technology file, no creation is necessary.

There is no special set-up required for this case.

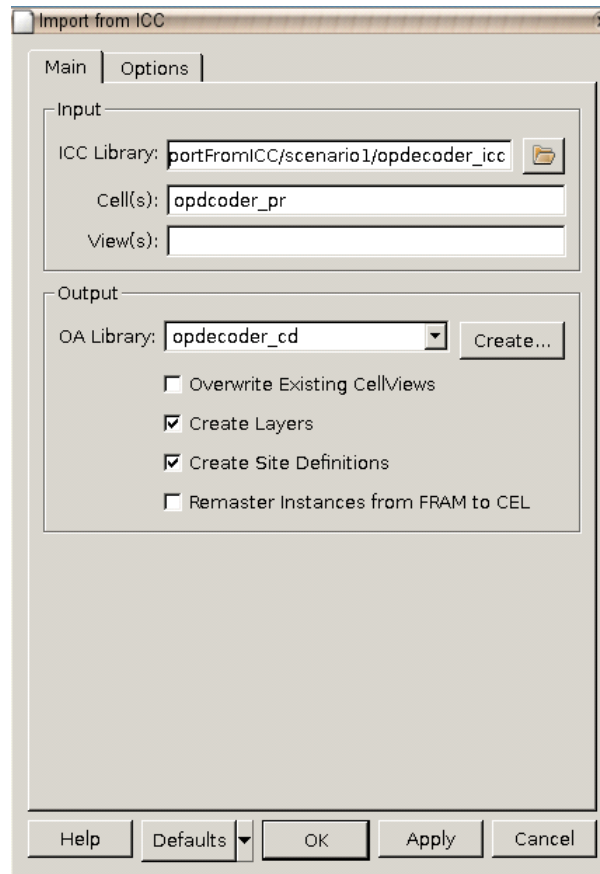
### **Example 5**

The IC Compiler library, `opdecoder_icc` has the metal1 layer (identified by the mask name "metal1") with layer name "m1" on layer number 8 with dataType 0.

The Custom Designer design library, `opdecoder_cd` with the Custom Designer tech database has the LPP "m1 drawing" on the LPP - 8:-1.

Similarly, the other layers also match.

When the IC Compiler design `opdecoder_icc/opdecoder.CEL` is imported to the Custom Designer library `opdecoder_cd`, the tech layer match is resolved by layer names.



*Figure 24 Import from ICC Dialog Box*

A layer mapping file is generated during the process. The layer mapping file is created and placed under the `iccbridge` directory. The file is called `layer.map`.

For the given example, the `layer.map` looks like the following:

CDLayerName	CDLayerPurpose	ICCLayerNumber	ICCDatatypeNumber
m1	drawing	8	0

---

## Scenario 3: Importing to an Existing Custom Designer Library with Different Layer Names

In this scenario, there is an IC Compiler library. There is also a Custom Designer library with a technology database available for import.

**Important:** In this case the layer names are mismatched between the IC Compiler and Custom Designer technology.

During the import process, for the IC Compiler layers on dataType 0 (which does not have a Custom Designer layer with same name), a new layer would be created in the Custom Designer technology database.

For the IC Compiler layers not on dataType 0, a new layer and purpose would be created if no other matching layer/purpose exists.

To avoid creating new layers/purposes during the import process, the required setup process would be to create a `layer.map` file to resolve the layer name mismatch.

### Example 6

The IC Compiler library `opdecoder_icc` has metal layers on M# (where #=1,..,10).

There are M#:M#DMY layer dataTypes that do not have an equivalent Custom Designer purpose.

The Custom Designer library has metal layers on m# (where #=1,..,10).

Table 10

IC Compiler Layers	Custom Designer Layers
M1 (8) : dataType 0	m1 drawing
M1(8): M1DMY(1)	???

As shown in [Table 10](#), all metal layers on dataType 0 map to the m# drawing LPPs on the Custom Designer side.

For the IC Compiler layers M#:M#DMY, there is no equivalent purpose on the Custom Designer side.

When the design `opdecoder_icc/opdecoder_pr.CEL` is imported into the Custom Designer library `opdecoder_cd`, using the default mapping (that is,

where no `layer.map` file is provided), then a new LPP "M1:M1DMY " would be created for IC Compiler layer M1:M1DMY.

On opening the `opdecoder_cd/opdecoder/layout`, the following messages are printed:

```
Information: create a new lpp(M1/drawing) with layer
number 10019 and purpose number -1 in tech lib
opdecoder_cd. (TECH-020)
```

```
Information: create a new lpp(M1/M1DMY) with layer number
10019 and purpose number 1 in tech lib opdecoder_cd.
(TECH-020)
```

To avoid creating a new LPP and using the existing layers, a `layer.map` file should be created.

The `layer.map` file must be used to resolve the mismatch, as shown below:

CDLayerName	CDLayerPurpose	ICCLayerNumber	ICCDatatypeNumber
m1	drawing	8	0
m1	M1DMY	8	1
m2	drawing	10	0
m2	M2DMY	10	1

## Appendix A: Setting the User Environment

### Import from the IC Compiler Tool

The `layer.map` file has to be specified in the Import From ICC dialog under the Mapping Tab as shown in [Figure 25](#):

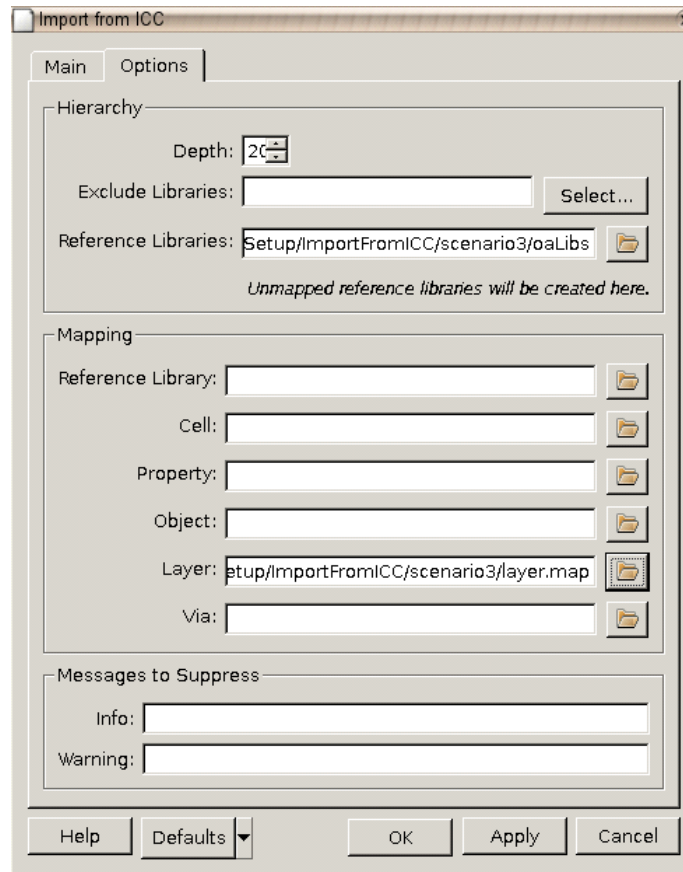


Figure 25 Import from ICC Dialog Box

The tool uses the specified layer mapping file and creates a new purpose M#DMY for each metal layer.

---

## Scenario 4: Importing to an Existing Custom Designer Library with No Matching Layers

In this scenario, there is an IC Compiler library. There is also a Custom Designer library with tech database that is available for import.

**Important:** In this case, there are some IC Compiler layer dataType elements that do not have an equivalent matching LPP in Custom Designer.



There are three methods that allow the creation of the missing layers/purposes in target Custom Designer technology:

- Use the automated behavior of the **Import From ICC** feature.
- Use the `layer.map` file
- Create a new technology file and editing the existing Custom Designer technology file to include the missing layers and purposes.

## Using the Automated Behavior of Import From IC Compiler

During the import process, the tool checks the IC Compiler layer names in the tech database against the Custom Designer's layer names.

- If the layer number is not already in use in the destination techfile, a new layer is created in the destination techfile with the source layer name and number.
- If the layer number is in use, a new layer is created in the destination techfile with the source layer name and an unused layer number.

The same process is applied for dataTypes, also.

- If the dataType number is not already in use in the destination Custom Designer techfile, a new purpose is then created in the destination techfile with the source dataType name and number. Otherwise, a new purpose created in the destination techfile with the source dataType name and an unused dataType number.

### Example 7

For the IC Compiler design library, `opdecoder_icc`, the design database has

- shapes on the Layer: dataType m#:dataType0 , where #=1,2,3 and 4
- dummy layers on the Layer: dataType m#:m#dummy, where #=1,2,3,4

There is already a Custom Designer library `opdecoder_cd`, which contains the equivalent metal layer definitions for layer: dataType m#: dataType, but no purpose definitions for the dataType m#dummy.

Table 11

IC Compiler Layers	Custom Designer Layers
m1 (8) : dataType 0	m1 (8) :drawing (-1)
m1(8):m1dummy(1)	m1(8): ???

## Appendix A: Setting the User Environment

Import from the IC Compiler Tool

Table 11

IC Compiler Layers	Custom Designer Layers
m2(10):dataType 0	m2(10) :drawing(-1)
m2(10):m2dummy(1)	m2(10):???
SCRB(88): dataType 0	???

When the IC Compiler design `opdecoder_icc/opdecoder.CEL` is imported into the Custom Designer library `opdecoder_cd`, the tech layer matching is resolved by the layer names.

On design open in Custom Designer, new LPPs are created for the missing layers and purposes and the messages below are output:

```
Information: create a new lpp(m1/m1dummy) with layer
number 8 and purpose number 10 in tech lib opdecoder_cd.
(TECH-020)
```

```
Information: create a new lpp(SCRB/drawing) with layer
number 88 and purpose number -1 in tech lib opdecoder_cd.
(TECH-020)
```

The generated layer map file looks as follows:

m1	drawing	8	0
m1	m1dummy	8	10
SCRB	drawing	88	0

The tech database is accordingly updated with the new LPPs as follows:

```
techPurposes (
; ( purposeName  purposeNum  abbreviation )
; ( -----      -----      -----      )
  ( m1dummy      10          )
) ;techPurposes
techLayers (
  ( SCRB          88          )
) ;techLayers
techLayerPurposePriorities (
  ( m1            m1dummy      )
  ( SCRB          drawing      )
...
) ;techLayerPurposePriorities
techDisplays (
  ( m1            m1dummy      m1_m1dummy      t t nil nil
t )
  ( SCRB          drawing      SCRB_drawing      t t nil nil
t )
)
```

## Using the Layer Mapping File

If you want to map the missing layers in the IC Compiler technology database to a specific layer number and purpose, you can provide an explicit mapping using the `layer.map` file.

If the destination layer number/purpose number provided in the `layer.map` file does not exist, then it is created.

### Example 8

For the example described in [Example 7](#), let us assume that your requirement is to map the missing layers as shown in the following table:

Table 12

Custom Designer Layers	IC Compiler Layers
m1 (8) :drawing (-1)	m1 (8) : dataType 0
m1(8): <i>dmy1(1)</i>	m1(8):m1dummy(1)
m2(10) :drawing(-1)	m2(10):dataType 0
m2(10): <i>dmy1(1)</i>	m2(10):m2dummy(1)

## Appendix A: Setting the User Environment

### Import from the IC Compiler Tool

Table 12

Custom Designer Layers	IC Compiler Layers
SCRB(88):drawing	SCRB(88): dataType 0

The `layer.map` file has to be created as shown below:

m1	dmy1	8	10
m2	dmy1	10	10
.....			
SCRB	drawing	88	0

The `layer.map` file has to be specified in the **Import From ICC** dialog under the **Options** tab as shown in [Figure 26](#):

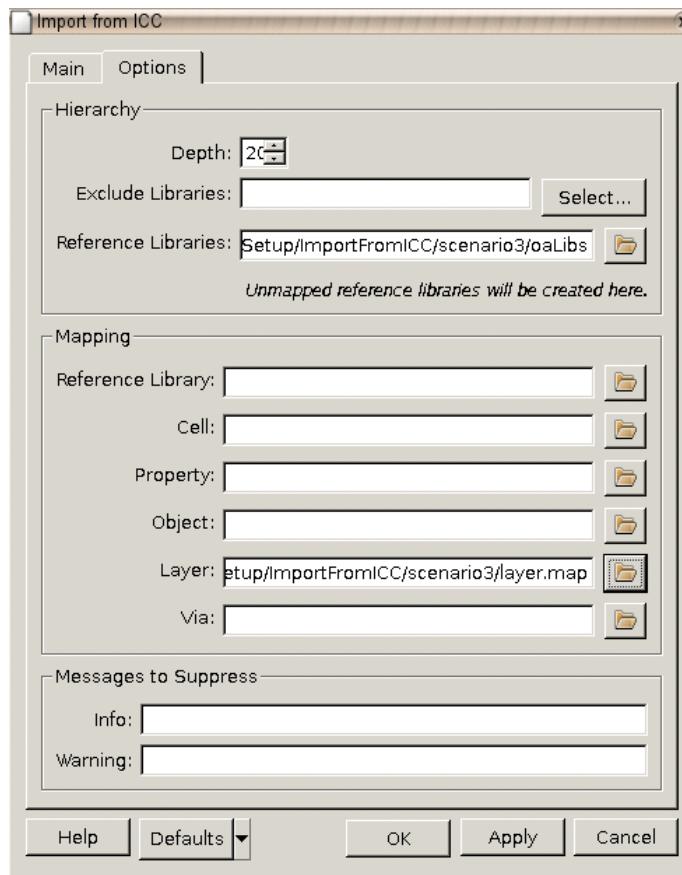


Figure 26 Import from ICC Dialog Box

## Scenario 5: Importing to an Existing Custom Designer Library with Read-Only Technology

In this scenario, there is an IC Compiler library. There is also a Custom Designer library with a tech database available for import.

In this case, there are some IC Compiler layer:dataType pairs that do not have a corresponding Custom Designer LPP and the Custom Designer technology database is read-only.

The set-up process required is to create an incremental technology file.

### Example 9

For the IC Compiler design library, opdecoder\_icc, the tech database and design database have:

- shapes on the Layer:dataType m#:dataType0 , where #=1,2,3 and 4
- dummy layers on the Layer:dataType m#:m#dummy where #=1,2,3,4

There is already an existing Custom Designer library opdecoder\_cd which contains the corresponding metal layer definitions for layer: dataType m#: dataType, but no purpose definitions for the dataType m#dummy.

Table 13

Custom Designer Layers	IC Compiler Layers
m1 (8) :drawing (-1)	m1 (8) : dataType 0
m1(8): ???	m1(8):m1dummy(1)
m2(10) :drawing(-1)	m2(10):dataType 0
m2(10):???	m2(10):m2dummy(1)
???	SCRB(88): dataType 0

When the IC Compiler design opdecoder\_icc/opdecoder.CEL is imported into the Custom Designer library opdecoder\_cd\_techreadonly, the import fails with the following message because there is no write permission to the Custom Designer technology database.

```
Error: Caught base exception: Error: Cannot open
OpenAccess library 'opdecoder_cd_techreadonly' with
'write' mode (MWBRIDGE-008) (MWBRIDGE-051)
```

## Appendix A: Setting the User Environment

Import from the IC Compiler Tool

An incremental technology file can be created to incrementally assemble technology information by creating references from one oaTech database to another oaTech database.

In this case, an incremental technology file has to be created using the `opdecoder_cd_techreadonly` as the base technology reference.

The incremental technology file would look as follows.

```
controls(  
  techArray(opdecoder_cd_techreadonly)  
) ;controls  
  
layerDefinitions(  
  
  techLayers(  
    ;( LayerName                Layer#                Abbreviation )  
    ;( -----                - - - - -                - - - - - )  
    ;User-Defined Layers:  
    ( SCRB                        88                        scrb          )  
  ) ;techLayers  
  techPurposes(  
    ( mldummy                    10                        )  
  ) ;techPurposes  
  
  techLayerPurposePriorities(  
    ;layers are ordered from lowest to highest priority  
    ;( LayerName                Purpose                )  
    ;( -----                - - - - -                )  
    ( m1                        mldummy                )  
    ( SCRB                      drawing                )  
  ) ;techLayerPurposePriorities  
  techDisplays(  
    ( m1                        mldummy                designFlow1      t t nil  
    nil t )  
    ( SCRB                      drawing                designFlow2      t t nil  
    nil t )  
  ) ;techDisplays
```

As shown above, the incremental technology file contains only the missing purpose and layers.

The display attributes are chosen to be the default system-defined display attributes and can be modified by you.

The next steps are:

1. Create a new `incr_tech` library using the above technology file `incr_tech.tf`.

2. Create a new top-design library `opdecoder_cd`.
3. Using the Technology Manager, set `incr_tech` library as tech reference for `opdecoder_cd` library, as shown in [Figure 27](#):

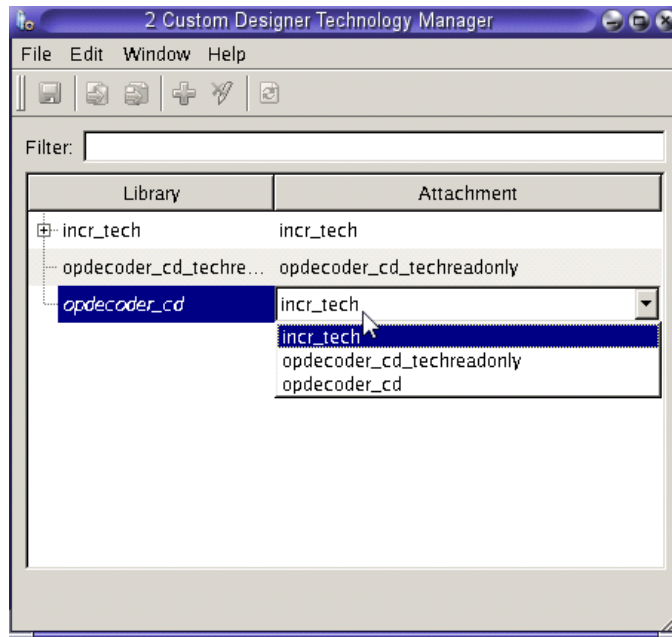


Figure 27 Technology Manager in Custom Designer

See [Technology Manager Tool](#) in the *Galaxy Custom Designer Platform User Guide* for more detailed information on using the Technology Manager tool.

4. Import the design `opdecoder_icc/opdecoder_pr.CEL` into the library `opdecoder_cd` and it should be successful.

**Note:** If there is layer name mismatch between the IC Compiler and Custom Designer technology, use a layer mapping file during the import process as explained in [Scenario 3: Importing to an Existing Custom Designer Library with Different Layer Names](#) on page 146.

---

## Pre-Route Flow

This section describes aspects of the pre-route flow in the following sections:

- [Creating Guard Ring Definitions for an IC Compiler Design in Custom Designer](#)
- [Using Separate OpenAccess Technology for an Opened IC Compiler Library](#)

---

### Creating Guard Ring Definitions for an IC Compiler Design in Custom Designer

This section covers two methods for creating Multi Part Path (MPP) guard rings on an IC Compiler design opened in Custom Designer through the Milkyway plug-in.

#### Example 10

The required setup is to create an incremental technology library with the MPP definitions, explained in the following example.

In this example, the requirement is to create an MPP definition for the IC Compiler design `opdecoder_icc/opdecoder_pr.CEL` in Custom Designer.

The required steps are:

1. Add and open the IC Compiler library on the Custom Designer platform.
2. Create an incremental technology file, which
  - a. references one of the IC Compiler reference libraries for base technology. (The example uses "stdcells" as the reference library for the top-design IC Compiler library `opdecoder_icc`.)

**Note:** When you create an incremental technology file, you will use "techArray" which is used as a tech pointer to mention the base technology data. This means that all the design constraints and process information in the tech pointer is applicable to incremental technology file.

As a result, when the techArray refers to the top IC Compiler design library, it will create a cyclic tech reference. To avoid the cyclic reference issue, one of the IC Compiler reference libraries is used as a tech pointer in techArray.



b. contains the MPP definitions.

The created incremental technology file (`incr_tech_mpp.tf`) looks as follows. A partial technology file is shown below for reference.

```
controls(
  techArray(stdcells)

  leMPPControls (
leMPPDefinition ( psubGR( ptypeGuardRing )          ( 1 ) 0 )

leMPPRingObject (
  ptypeGuardRing defaultPath
  ( diffEncPath pplusEncPath )
  nil
  ( defaultContacts )
  nil
  VSS
  userParams( name netName )
);leMppRingObject

leMPPDefinition ( nwellGR( ntypeGuardRing )          ( 1 ) 0 )

;leMPPRingObject (
leMPPRingObject (
  ntypeGuardRing defaultPath
  ( diffEncPath nplusEncPath nwellEncPath )
  nil
  ( defaultContacts )
  nil
  VDD
  userParams( name netName )
);leMppRingObject
.....<<Some of the sections skipped>>.....
) ;leMPPControls
) ;controls
```

For more detailed information, see [leMppDefinitions](#) in the *Custom Designer Technology and Display Resource File Reference Manual*.

The next steps are:

1. Create a new `incr_mpp_tech` library using the above technology file `incr_mpp_tech.tf`.
2. Set this `incr_mpp_tech` library as tech reference for `opdecoder_icc`, as shown in [Figure 28](#).

## Appendix A: Setting the User Environment

### Pre-Route Flow

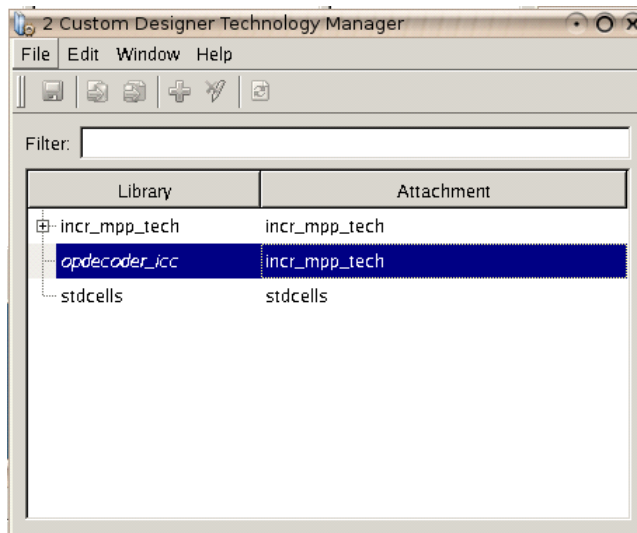


Figure 28 Technology Manager in Custom Designer

3. Open the `opdecoder_icc/opdecoder_pr/layout`.
4. Choose **Create > Multi Part Path > Manual** or **Automatic** mode to create the MPPs.

### Example 11

The following steps describe another method to create MPPs in an IC Compiler design. Note that this example applies to adding layer definitions; alternatively, a Custom Designer technology library can also be used as a technology reference.

1. Create a custom technology file with MPP definitions.
2. Include any MPP layers that are not defined in the IC Compiler library.
3. Import the technology file in the IC Compiler library using Merge mode (shown in [Figure 29](#)).

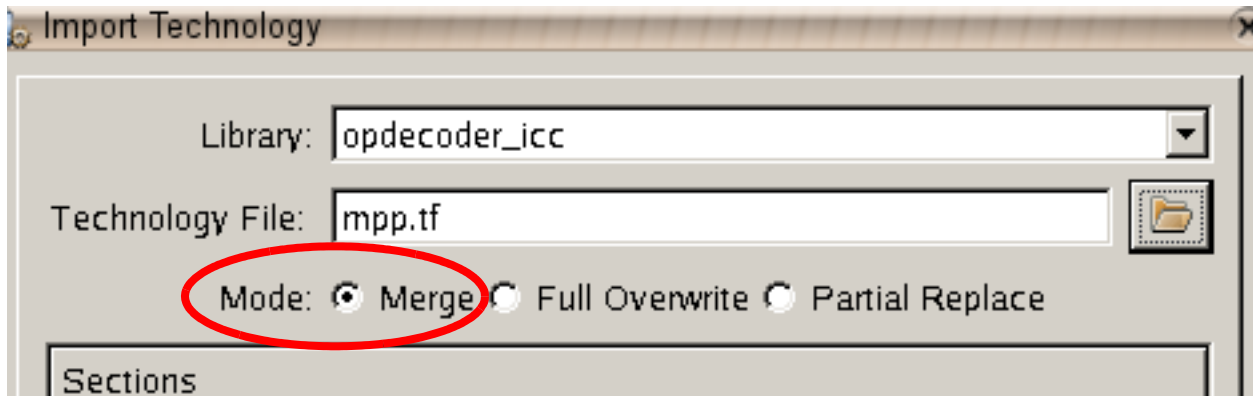


Figure 29 Merge Mode

4. Save the IC Compiler library technology after import (shown in [Figure 30](#)).

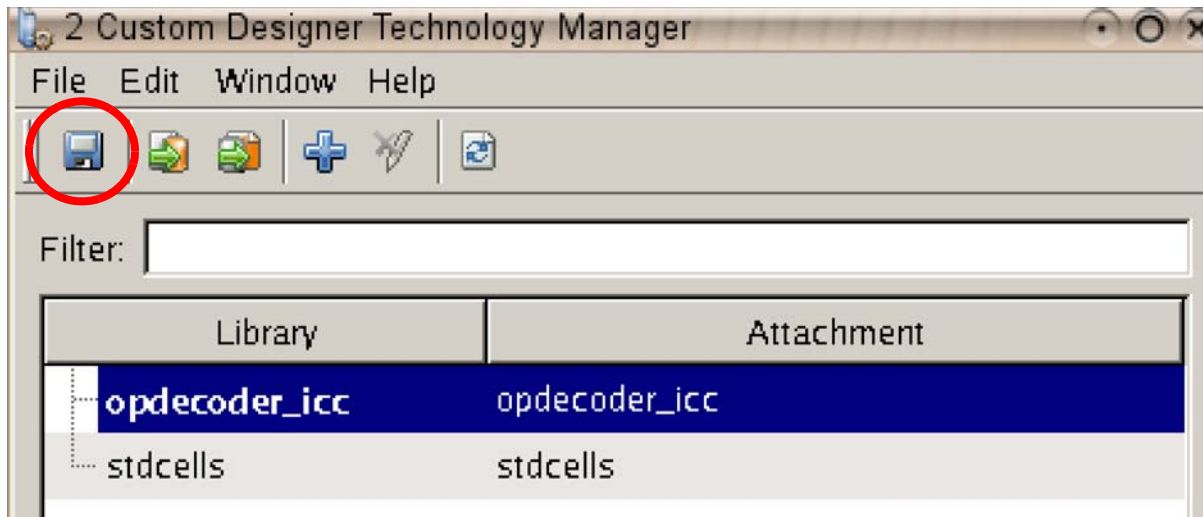


Figure 30 Save the Technology File

---

## Using Separate OpenAccess Technology for an Opened IC Compiler Library

This section covers using an OpenAccess technology file for an IC Compiler design opened in Custom Designer through the Milkyway plug-in.

To use a Milkyway library in the context of a technology defining Custom Designer, the required set-up is:

- Use an OpenAccess technology attachment, which disables technology synchronization; and
- (Optional) Use mapping files if necessary to reconcile differences between layer and via names defined in OpenAccess versus those defined in Milkyway.

**Note:** Using a separate OA technology requires write access to the IC Compiler library.

The set-up is explained in the following example.

### Example 12

In this example, the layer names are mismatched between the Custom Designer and IC Compiler technology databases. The requirement is to view the IC Compiler design in the context of the OpenAccess technology file.

Table 14

---

Custom Designer Layers	IC Compiler Layers
m1 (8) :drawing (-1)	M1 (8) : dataType 0
m2(10):drawing (-1)	M2(10): dataType 0
.....	
m10(85):drawing(-1)	M10(85):dataType 0

---

The required steps are:

1. Set this `oa_tech` library as tech reference for `opdecoder_icc`, as shown in [Figure 31](#).

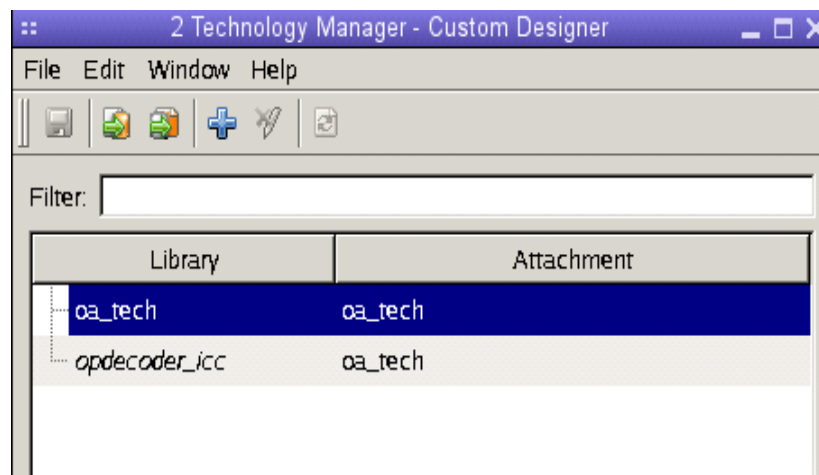


Figure 31

2. The `layer.map` file used to resolve the mismatch is shown below:

CDLayerName	CDLayerPurpose	ICCLayerNumber	ICLayerDatatypeNumber
m1	drawing	8	0
m2	drawing	10	0
m3	drawing	12	0
m4	drawing	14	0
m5	drawing	70	0
m6	drawing	72	0
m7	drawing	74	0
m8	drawing	76	0
m9	drawing	78	0
m10	drawing	85	0

3. Disable the synchronization using the `dm::setICCTechSync` command:

```
set lib [dm::getLibs opdecoder_icc]
dm::setICCTechSync $lib -layerMap layer.map -sync false
```

## Setup Using LEF Files

This section explains how to

- Create an IC Compiler library from the LEF files description available for the technology and standard cells
- Create an OA library from the LEF files description available for the technology and standard cells

---

### Scenario 1: Importing LEF Files to Create an IC Compiler Library

The Milkyway tool can be used to read the LEF file to create IC Compiler technology file and IC Compiler libraries.

For further details on the flow, see the IC Compiler SolvNet article, "[Library Preparation Using LEF/DEF, Version F-2011.09.](#)"

#### Example 13

In this example, there are two LEF files available:

- `cells.lef`, which contains the descriptions for the standard cells.
- `tech.lef`, which contains the technology information.

In Milkyway, use the `read_lef` command to create an IC Compiler technology file and library from the above LEF files:

1. Invoke Milkyway in Tcl mode:

**UNIX% Milkyway -tcl &**

2. In the Milkyway window, run the command to create an IC Compiler library `stdcells`:

**read\_lef -lib\_name stdcells -tech\_lef\_files tech.lef -cell\_lef\_files cells.lef**

3. Dump the IC Compiler technology file for the above `stdcells` library by choosing **Tech File > Write to File**.

---

## Scenario 2: Importing LEF files to create a Custom Designer library

To create Custom Designer libraries, the `lef2oa` standalone translator can be used to import the LEF files.

### Example 14

In this example, there are two LEF files available:

- `cells.lef`, which contains the descriptions for the standard cells.
- `tech.lef`, which contains the technology information.

To create a Custom Designer technology file and library from the above LEF files, use the `lef2oa` standalone command.

To import the LEF, a layer mapping file is required to map LEF objects found on the specified `layerName` to the specified `layerNumber`.

If a layer mapping file is not specified, the layer names are auto-generated.

The following is an example of a layer mapping file to use:

LayerName	Layer Number
poly	7
m1	9
m2	11
m3	13
m4	15
m5	17
m6	19
m7	21
m8	23
m9	25
m10	27

1. In the UNIX window, type the following command to create an OA library `stdcells_oa` and a tech library `tech_lib`:

```
lef2oa -lib stdcells_oa -lef tech.lef -layerMap layer.map -techLib  
tech_lib
```

The above command creates a new cell library called `stdcells_oa`. The option `techLib` specifies the name of a technology library to use when creating a new cell library. If the specified technology library does not exist, the library is created using the technology information in the LEF file.

2. In the UNIX window, type the following command to create an OA library `stdcells_oa` and a tech library `tech_lib`.

## Appendix A: Setting the User Environment

### Setup Using LEF Files

```
lef2oa -lib stdcells_oa -lef cells.lef -layerMap layer.map -techLib  
tech_lib
```

The above command imports the cell information from the `cells.lef`. During import, it uses the technology information from `tech_lib` created in [Step 1](#).



## C

cell.map, ICC mapping file, table 30

## D

design view types and documentation, table 46

## F

FRAM views, generating 2

## I

IC Compiler

- adding ICC library access 51

- design library

  - export 34

  - import 25

- library mapping files 30

- target library import 31

ICC FRAM views, generating 2

iccbbridge run directory 29, 40

## L

layer.map, ICC mapping file, table 30

lib.map, ICC mapping file, table 30

## O

object.map, ICC mapping file, table 31

## P

parameter

- definition 20, 22

property.map, ICC mapping file, table 31

## R

route object 74

## V

via.map, ICC mapping file, table 31

