Laker User Guide and Tutorial

Laker Custom IC Design Solutions

Synopsys, Inc.

Printing

Printed on August 22, 2013.

Version

This manual supports the Laker™ Custom Layout Automation System 2013.08 and higher versions. You should use the documentation from the version of the installed software you are currently using.

Copyright and Proprietary Information Notice

Copyright (c) 2013 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.

Certain software components distributed with Laker are licensed under the GNU General Public License, Version 2 (GPL), or GNU Lesser General Public License, version 2.1 (LGPL). The software licensed under the GPL and LGPL is free software that comes with ABSOLUTELY NO WARRANTY. You may modify, redistribute, and otherwise use the GPL and LGPL software under the terms of the GPL (http://www.gnu.org/licenses/gpl-2.0.html) and LGPL (http://www.gnu.org/licenses/lgpl-2.1.html), respectively. Synopsys does not own or control these websites. Synopsys neither evaluates nor makes any representations regarding the accessibility of these websites.

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

Contents

About This Book

Overview

This book is designed to allow you to quickly become proficient in the Laker™ Custom Layout Automation System, the most intelligent full custom layout design system.

The book should be read from beginning to end. Sections you are already familiar with can be skipped.

Audience

The audience for this book includes physical layout engineers who require faster and more efficient automated layout design tools.

This document assumes that you have a basic knowledge of the platform on which your version of the Laker layout system runs: Unix or Linux.

Book Organization

This *Laker User Guide and Tutorial* is organized as follows:

- "About This Book," provides an introduction to this book and explains how to use it.
- "Introduction," provides an introduction to the Laker™ Custom Layout Automation System.
- "User Interface," describes the graphical user interface (GUI), common features, selection schemes, and primary windows.
- "Quick Start," provides more practice and helps you to skillfully exploit the advanced functionalities of the Laker layout system.
- "UDD Tutorial," provides more practices on user-defined devices (UDDs).
- "Index," is a detailed index to this book.

Conventions Used in This Book

The following conventions are commonly used in this book:

- *Italics* font is used for emphases, book titles, section names, design names, file paths, and file names within paragraphs.
- **Bold** font emphasizes text and highlighted titles, menu items, function keys, and button names.
- *Blue text* outlines the text area for active hypertext links; it helps you jump to the reference topic.
- `Courier type` is used for program listings, Tcl commands, and arguments. It is also used for test messages that appear on the screen.
- **NOTE** describes important information, warnings, or unique commands.
- Left-click or Click means click the left mouse button on the indicated item.
- Middle-click means click the middle mouse button on the indicated item.
- Right-click means click the right mouse button on the indicated item.
- Double-click means click twice consecutively with the left mouse button.
- Shift-left-click means press and hold the **Shift** key then click the left mouse button on the indicated item.
- Shift-right-click means press and hold the **Shift** key then click the right mouse button on the indicated item.
- Drag-left means press and hold the left mouse button, then move the cursor to the destination and release the button.
- Drag-right means press and hold the right mouse button, then move the cursor to the destination and release the button.
- Drag and drop means press and hold the middle mouse button on the indicated item then move and drop the item to the other window.

Functions of Common Buttons

This section describes the most common buttons that appear in GUI forms.

- The **OK** button confirms the settings in the form and closes the form.
- The **Apply** button applies any changes but does not close the form.
- The **Cancel** button discards the settings in the form and closes the form.
- The **Close** button closes the form without applying any changes.
- The **Hide** button temporarily hides the form during operation. It can be brought back again by pressing **F3**.

Related Publications

- *Laker Command Reference Manual* - gives detailed information on the Laker command sets, with a reference of available bind keys, toolbar icons, and links to Tcl commands.
- *Laker Tcl Reference Manual* - explains the use of Tcl commands for primary windows and introduces the enhanced Tcl commands to access the GUI and database.
- *Laker Automated Test Chip Development Platform User's Manual - g*ives detailed information on the Laker Automated Test Chip Development (TCD) Platform command sets, with the reference of available bind keys, toolbar icons and links to Tcl commands.
- *Laker Automated Test Chip Development Platform Tcl Reference Manual* - gives detailed information on the Laker Automated Test Chip Development (TCD) Platform Tcl commands.
- *Laker Quick Reference Guide* - gives a brief summary of the different windows and related mouse commands and bind keys.
- *Installation Guide* - gives detailed information on installing the Laker™ Custom IC Design products.
- *Release Notes* - For current information about the latest software version, see the *Release Notes* shipped with the product.

Introduction

Overview

The Laker^{TM} Custom Layout Automation System offers powerful solutions for analog, mixed-signal, memory, and custom digital IC design that address key pain points in the layout process. The Laker layout system provides an intuitive methodology and simplifies the layout process by automating many tedious and error-prone layout tasks.

When the Laker layout system is deployed in a design flow, less effort is required to obtain optimal, high-quality, high-density layout of advanced chip designs. Superior layout results are achieved by:

- Speeding up the project and optimizing the layout at the same time;
- Efficiently interacting with the netlist, schematic and layout data in a single working environment;
- Cutting overall DRC/LVS run times by creating layout that is DRC-correct the first time; and
- Reducing or eliminating PCell scripting with unique device generation technologies.

The Laker layout system with its controllable automation technologies simplifies the layout process and reduces physical verification efforts by offering rule-driven and schematic-driven capabilities and an open architecture that seamlessly integrates best-in-class third-party tools.

Technology Overview

At the core of the Laker layout system's controllable automation is Synopsys' patented Magic Cell (MCell^{TM}) parameterized device generation technology and a built-in design rule check (DRC) engine that drives the system's rule-driven layout capabilities. MCells enable a highly automated schematic-driven layout (SDL) flow that includes the user-controllable device planning, wiring and manipulation capabilities of the Stick Diagram Compiler.

- MCells include transistors, resistors, capacitors, contacts/vias and guard rings.
- MCells enable the rapid generation of optimized layout structures for complex devices, such as multiple-gate transistors, guard-rings, contact arrays, inter-digitized resistors and capacitors.
- MCells are technology-independent and require no user scripting.

Interoperability

The Laker Custom Layout System provides unparalleled interoperability and an open environment that employs best-in-class design tool flows that include, physical verification, extraction and much more. This open system provides:

- Seamless integration with Calibre^{R} and Hercules^{TM} physical verification tools that executes DRC and LVS from the Laker window and supports easy browsing and debugging of the results; and
- Complete Tcl support including language syntax and selected extensions to provide maximum flexibility in tool customization, database query, and layout creation.

Product Overview

The Laker layout system includes the following features.

Core Features

The Laker layout system provides rule-driven layout so physical layout that is DRC and LVS correct can be rapidly realized, placed, routed, and edited. Unique controllable automation simplifies numerous tasks from measurement to device generation. The built-in DRC engine drives the application of design rules through device generation, layout, and routing.

Rule-driven Layout

Automatically checks, displays, and snaps to width, space, notch, overlap, and enclosure rules in real time. The Laker layout system supports:

- Selecting an area of layout and automatically correcting DRC violations.
- LVS-correct layout results with flight lines that display connectivity information to guide and speed-up wiring operations.
- Automatically identifying any created shorts in real time.
- Moving same-layer routes out of the way when hand-routing high priority nets with the Push Wire feature.
- Rules for width, space, notch, overlap, and enclosure including foundry-recommended and DFM rules.

Built-in Routers

The built-in routers in the Laker layout system use the design's connectivity to automatically finish wiring "on the fly". Features include:

- A point-to-point router that either automatically or interactively creates a DRC-correct route between the source and target.
- Interactive, DRC-correct Pathfinder that follows the cursor in "point and click" routing mode, recognizing and routing around same layer metal - or switch layers with the click of a bind key and Pathfinder will retain connectivity by automatically placing the appropriate via stacks.
- A route-by-label function that automatically creates routes between multiple points as guided by text or labels.
- A net router that automatically routes single or multiple nets.

User-Defined Device (UDD)

For complex devices, such as spiral inductors, the Laker layout system offers a convenient tool that creates parameterized devices (similar to PCells) without scripting. A device can be drawn in the UDD layout window, and then layout rules and parameters can be set using the built-in form, and parameters to be changed by users can be assigned. In addition, existing layout, including legacy PCell layout, can be read into the UDD tool and parameterized for use in the Laker layout system.

Advanced Features

The Laker layout system uses unique technologies to exploit design rules, connectivity and parameters during layout in an efficient, consistent, and automatic way.

Schematic-driven Layout

With schematic-driven layout, optimized layout can be rapidly created that is DRC/LVS-correct in less time without sacrificing density or design styles. Both netlist and schematic views are included with the Laker layout editor for an intuitive SDL working environment, which includes:

- A hierarchical design browser for rapidly searching the hierarchy and cross- probing between schematic, netlist, and layout. The design hierarchy can be modified for more efficient layout by flattening logical groups of devices into higher level devices.
- A schematic view for enabling schematic-driven layout. Cross-probe with the design browser to identify candidates for flattening. Automatically identify repeated patterns, associate optimized layout for those patterns and generate layout in seconds.
- A layout editor for rapid, rule-driven layout. Realize, place, route, and edit physical layout that is DRC- and LVS-correct.
- A patented automatic schematic generator for creating a readable schematic from CDL or SPICE netlists.

Stick Diagram Compiler

Based on the unique MCell technology, the Stick Diagram Compiler executes transistor floor-planning on-the-fly during device generation in the SDL flow. View and optimize device layout at a higher level of abstraction: swap, merge, move, split and align gates at a symbolic level without having to worry about design rules, connectivity or parameter values. The built-in automatic transistor placer offers a multi-row placement capability for PMOS and NMOS transistors that displays the best transistor placement for area and routing based on the given placement criteria.

Matching Device Creator

The Matching Device Creator is similar to the Stick Diagram Compiler, but custom-tailored to create matching device layout. It is also used to create resistors and capacitors.

Design Hierarchy Manipulation for Layout Optimization

For a more efficient layout implementation, a pattern can be created by grouping multiple circuit elements into a higher level device, and then you can automatically locate the pattern throughout the circuit and associate all occurrences of it with the optimized layout generated for the pattern while retaining full connectivity. Manipulation of circuit hierarchy in the design browser and layout windows drastically reduces the time spent laying out repeat circuitry.

Hierarchical Net Tracer

Using the relationships defined in the technology file section, the Hierarchical Net Tracer interactively traces the connection relationship of a net. The Hierarchical Net Tracer supports cross window viewing, creating and editing to a user-specified tracing depth.

User Interface

Overview

The Laker™ system is a multi-window application with an easy-to-use graphical user interface (GUI). It runs under Unix and Linux.

The Laker system has a number of commands, including many that you invoke through mouse clicks or drags rather than selecting from pull-down menus at the top of each window. Read this chapter to familiarize yourself with the interface conventions of the Laker system before you proceed further.

This chapter covers the following topics:

- *Common User Interface Features*
- *Main Window User Interface*
- *Design Window User Interface*
- *Selection Schemes*

Common User Interface Features

This section describes the common user interface features in the *Main* and *Layout* windows.

Window Banner

The banner at the top of each window may vary depending on the check-out license.

Main Window Banner

The banner at the top of the *Main* window specifies the application name with its license level. When using the L3 license, the window banner of the *Main* window will look like the following figure:

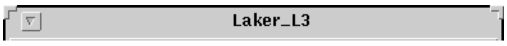


Figure: Main Window Banner

The banner of the *Main*, *Layout*, and *Schematic* windows can be customized to show extra information by defining one of the following settings before the Laker software is started:

- Set a keyword `WndUserBannerTitle` in the `[LeoPreference]` section of the *laker.rc* resource file. For example:

  ```
  [LeoPreference]
  WndUserBannerTitle = "90nm Project X"
  ```

- Set an environment variable `WndUserBannerTitle` in UNIX. For example:

  ```
  setenv WndUserBannerTitle "Project X"
  ```

  Once the environment variable is set, the value in the *laker.rc* resource file will be ignored.

The *Main* window banner with extra information will look like the following figure:

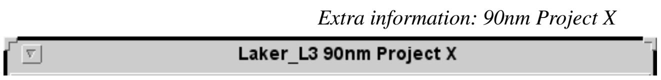


Figure: Main Window Banner with Extra Information

Layout Window Banner

The banner at the top of the *Layout* window identifies the application name, license level, window number, library name, cell name, user unit, open mode, view level, and the saved or unsaved status of the layout.

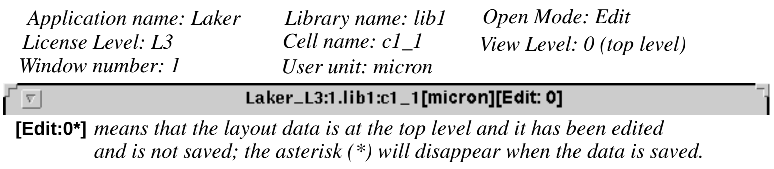

Application name: Laker
License Level: L3
Window number: 1
Library name: lib1
Cell name: c1_1
User unit: micron
Open Mode: Edit
View Level: 0 (top level)
Laker_L3:1.lib1:c1_1[micron][Edit: 0]
[Edit:0*] means that the layout data is at the top level and it has been edited and is not saved; the asterisk (*) will disappear when the data is saved.

Figure: Layout Window Banner

For example, `Laker_L3:1.lib1:c1_1[micron][Edit:0*]` means that Laker is started with the L3 license, and this is the first window since it has been started. This window banner also tells the user unit of the layout, and indicates that cell `c1_1` from library `lib1` has been opened with Edit mode at the top level and the layout has been edited and is not saved at the moment. The asterisk (`*`) will be removed when the layout is saved.

The display of the user unit (defined in the technology file section `tfLayoutSystemUnit`) on the window banner can be set in the *laker.rc* resource file designated by the keyword `DsgWndUnitOnBanner`.

The *Layout* window banner with extra information will look like the following figure:

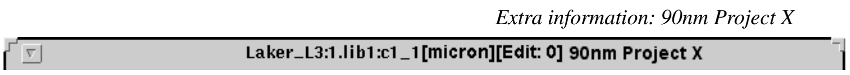

Extra information: 90nm Project X
Laker_L3:1.lib1:c1_1[micron][Edit: 0] 90nm Project X

Figure: Layout Window Banner with Extra Information

Schematic Window Banner

The banner at the top of the *Schematic* window identifies the application name, license level, window number, library name, cell name, and open mode.

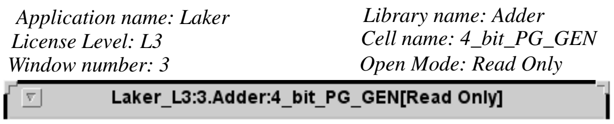


Figure: Schematic Window Banner

The *Schematic* window banner with extra information will look like the following figure:

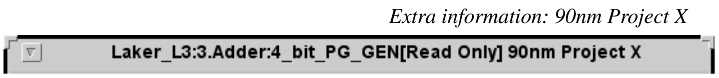

Extra information: 90nm Project X
Laker_L3:3.Adder:4_bit_PG_GEN[Read Only] 90nm Project X

Figure: Schematic Window Banner with Extra Information

Pull-down Menus

A pull-down menu bar is located just below the window banner. Each menu item contains several commands that are displayed when the menu is selected. A command can be invoked by choosing it from the pull-down menu or by using the available bind key.

The Laker system supports the tear-off menu convention, which allows a frequently used menu to be pulled aside and left open.

The appearance of pull-down menus can be customized by modifying the menu files. Refer to the *Customizing Laker* description in the *Resource File and Customization* chapter of the *Laker Command Reference Manual* for more details.

Mnemonic Keys

The window menus support **Meta** key invocation using mnemonics. The mnemonic for each item is indicated by an underline (_).

For example, to display the **File -> Open** menu (meta -fo), press and hold the **<Meta>** key on your keyboard (the diamond key/**<Alt>** key on Sun keyboards, or the <Alt> key on Windows' keyboards) and press the **f** key, and then release the **<Meta>** key and press the letter **o** key.

Bind Keys

You can bind a command to a keystroke. After you have defined bind keys, you can easily invoke commands with a keystroke.

Refer to the *Bind Key Summary* for the available bind keys in the Laker system.

Refer to the *Customizing Laker* description in the *Resource File and Customization* chapter of the *Laker Command Reference Manual* for more details on the customization of bind keys.

Esc Key

Pressing the **Esc** key can abort a command if the options form does not appear.

If the form appears, pressing the **Esc** key can close the options form or the object attributes form, without aborting the command (the command remains executed).

On-line Help

All application windows provide hypertext-based on-line help, which can be accessed through the **Help** menu on the main toolbar.

Pressing the **F1** key within an active options form will open *a quick access menu* to the command description in the *Laker Command Reference Manual*. This index-like menu is organized by the appearance of commands in each pull-down menu.

Toolbar Icons

The convenient toolbar icons appear beneath the pull-down menu bar. These toolbar icons give you access to frequently-used commands at the current window.

Mouse Operations

The available mouse operations are described in the following paragraphs.

Mouse Operations for Selection

The mouse is most often used to select objects, edges or vertex in the layout by clicking the left mouse button.

You can select ranges of objects by dragging with the left mouse button over the objects. For example, to select a region in the layout, use **Shift+a** to invoke the **Area Select** command, and then drag-left to select a region.

You can also select a range by using the **Shift** key while left-clicking. For example, to select multiple objects, edges, or vertices, use the **Shift** key while left-clicking.

To add or remove individual objects to or from the selection, use the **Ctrl** key while left-clicking.

To deselect all objects, simply left-click on an empty area in the layout.

Refer to the Selection Schemes section for details.

Mouse Operations for Creation

When drawing rectangles, circles or ellipses, left-click to start or complete the drawing operation.

When drawing polygons or paths, left-click to start drawing or add a vertex for these shapes, and then right-click to complete the drawing operation.

When drawing instances or MCells, right-click to rotate an instance or MCell 90 degrees counterclockwise before placing the device; Ctrl+right-click to turn an instance or MCell upside down before placing the device; Shift+right-click to mirror an instance or MCell before placing the device.

When drawing doughnuts, left-click to place the center of the doughnut, and draw the first and second circles of the doughnut.

Mouse Operations for Editing

When the **Move** or **Stretch** command is active, you can move selected objects or stretch selected edges or vertices by dragging with the left mouse button over the objects, edges or vertices.

Mouse Operations for Zooming and Panning

To zoom in an area in the layout, use **Ctrl+z** to invoke the **Zoom In** command, and then left-click in the layout area. If the command is not invoked, you can also drag-right on the layout area to perform the zoom in operation.

To show or hide the content window in the layout pane, middle-click in the layout area.

To pan to a selected point, use **Shift+Tab** to invoke the **Pan** command, and then left-click to perform the panning operation.

Mouse Operations in the Layout Table Pane

Refer to the *Layout Table Pane* description for more details on the mouse operations in the *Layout Table* pane of the *Design* window.

Right-click Command Menus

Right-clicking a library or cell in the *Open Cell* form shows a right-click command menu, which includes the commands to access that library or cell. Refer to the *Open Cell* description in the *Main Window* chapter of the *Laker Command Reference Manual* for more details on the right-click command menus of **Library** and **Cell** commands.

Right-clicking an instance in the *design hierarchy browser* pane shows a right-click command menu. Refer to the *Right-click Command Menu* description in the *Design Hierarchy Browser* chapter of the *Laker Command Reference Manual* for more details.

Right-clicking in the *Schematic* window shows a right-click command menu, which access the **Schematic** commands in the *Layout* window pane and the commands in the *design hierarchy browser* pane. Refer to the *Right-click Command Menu* description in the *Schematic Window* chapter of the *Laker Command Reference Manual* for more details.

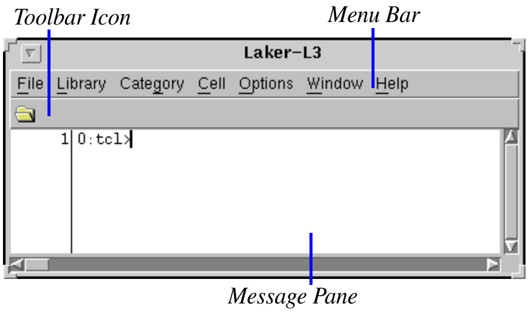
Right-clicking in the *Layout* window pane shows a right-click command menu, which access the **Move**, **Copy**, **Delete** and **Attribute** commands. Refer to the *Defining Right-click Command Menu* description of the *Resource File and Customization* chapter in the *Laker*

Command Reference Manual for more details on how to re-define the right-click command menu in the *Layout* window pane.

Main Window User Interface

The *Main* window is displayed after Laker is started. From this window, you may convert several external data types, such as GDSII/ASCII, DEF/LEF and CDL netlist, to the Laker database, and vice versa.

After the data is imported, you may further build up the data structure, such as creating or classifying the libraries and cells, configuring the technology file and library path mapping, and defining the favorite settings to ease the layout tasks in the Laker system.



```
1 0:tcl>
```

Figure: Main Window

The *Main* window consists of the following key components.

Toolbar Icon

There is only one toolbar icon in the toolbar of *the Main* window. When you rest your mouse cursor over the toolbar icon, you will see a yellow tip window, indicating the function of the selected button.

Clicking the **Open Cell** toolbar icon will bring up the *Open Cell* form, for which you need to provide library name, cell name, and search path information in order to open a cell view.

Message Pane

A message pane appears beneath the toolbar. From this message pane, you can check the messages associated with system warnings, data conversion, executed commands, and preference settings. A log file (*laker.log*) is provided to record the messages displayed in this pane.

On the other hand, you may execute the enhanced Tcl commands in this pane. For example, you can type `lakerOpenDesign -lib ram -cell ram -mode edit` and then hit the **Enter** key. The layout view of the cell (`ram`) in the library (`ram`) will be opened in the *Design* window.

Refer to the *Laker Tcl Reference Manual* for more details on available Tcl commands.

Design Window User Interface

The *Design* window is displayed after a cell is opened. This window consists of a pull-down menu bar, two rows of convenient toolbar icons, a comprehensive status bar and the following window panes: *Layer Table*, *Content* window, *Layout* window, *Design Hierarchy Browser*, and *Schematic* window.

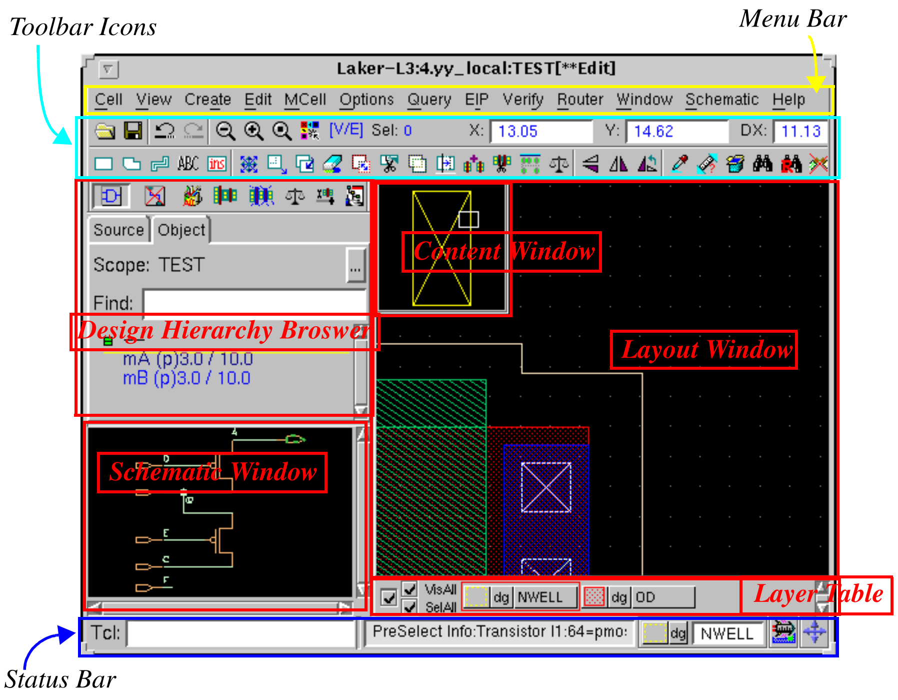

Toolbar Icons
Menu Bar
Laker–L3:4.yy_local:TEST[**Edit]
Cell View Create Edit MCell Options Query EIP Verify Router Window Schematic Help
[V/E] Sel: 0 X: 13.05 Y: 14.62 DX: 11.13
Source Object
Scope: TEST
Find:
Design Hierarchy Broswer
mA (p)3.0 / 10.0
mB (p)3.0 / 10.0
Content Window
Layout Window
Schematic Window
VisAll SelAll dg NWELL dg OD
Layer Table
Tcl:
PreSelect Info:Transistor I1:64=pmo:
dg NWELL
Status Bar

Figure: Design Window

These key components of the *Design* window are described in the following sections.

Toolbar Icons

The following figures show the toolbar icons available in the *Design* window and *design hierarchy browser* pane.

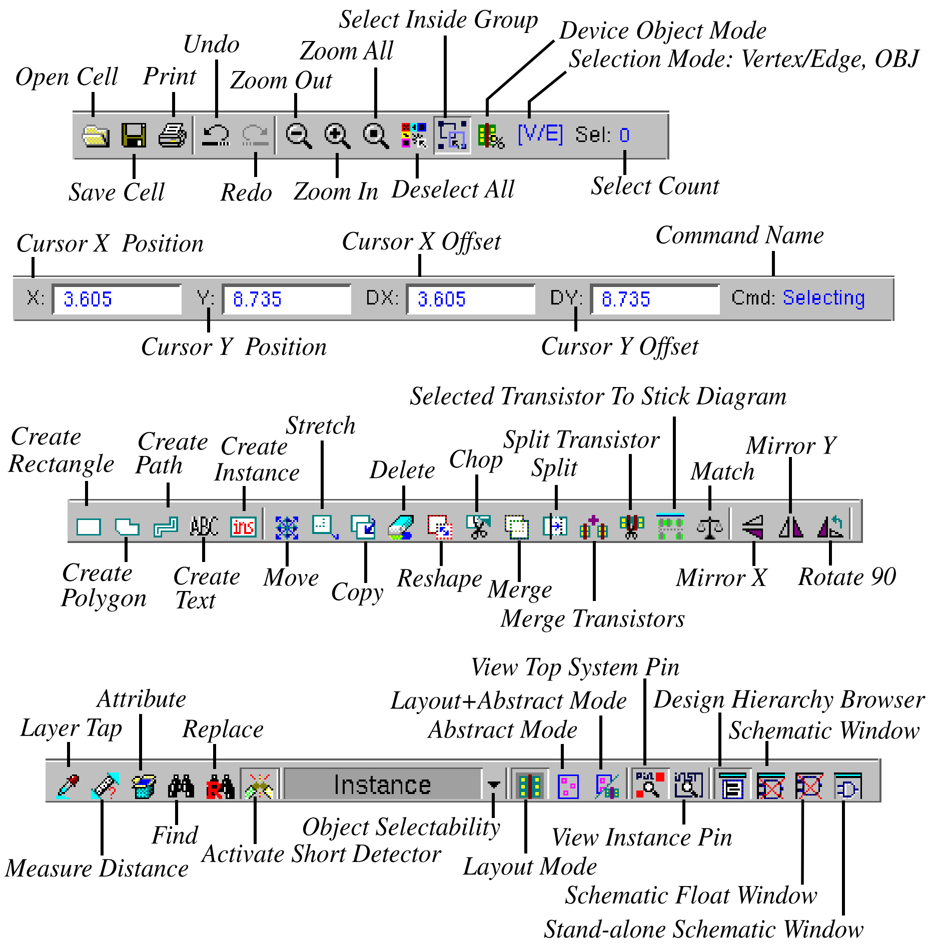


Figure: Design Window Toolbar Icons

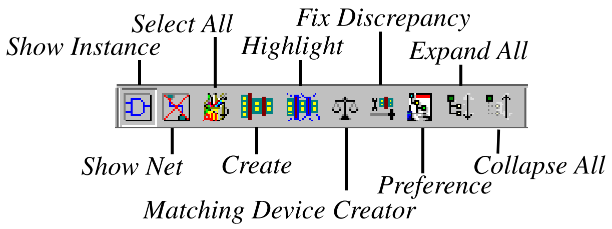


Figure: Design Hierarchy Browser Toolbar Icons

Layout Window Pane

The *layout* window pane is the window where the layout drawing is performed. You can open multiple *Design* windows and work in them simultaneously.

Refer to the *Layout Window* chapter of the *Laker Command Reference Manual* for more details.

Design Hierarchy Browser Pane

The *design hierarchy browser* pane appears, in the upper left of the *Design* window, only when opening a schematic cell view or a layout cell that contains the schematic data.

Refer to the *Design Hierarchy Browser* chapter of the *Laker Command Reference Manual* for more details.

Schematic Window Pane

The *Schematic* window pane appears only when you open a schematic cell view or a layout cell that contains schematic data. This window presents a schematic view of your netlist data in initial or revised state.

Refer to the *Schematic Window* chapter of the *Laker Command Reference Manual* for more details.

Content Window Pane

To zoom in/out of different layout areas, drag the right mouse button in the *Layout* window pane to specify an area on the *Content* window pane. The yellow rectangle presents an outline of a full opened cell, and the white rectangle presents the current view area.

The content window pane can be shown or hidden by clicking the middle mouse button in the *Layout* window pane. This content window pane is hidden by default.

Layer Table Pane

The *Layer Table* pane contains layers that are used to create objects.

Each layer has a different color/pattern and name, ensuring that it stands out against multiple layers. A yellow tip window, showing *layerName*, *layerNo*, and *PurposeName*, is displayed when the mouse cursor is placed over a selected layer. The layer outlined in red indicates that it has been selected.

Each layer comprises two parts: Visible Control Part and Select Control Part. The former presents the visible attribute (Layer Color/Pattern) and the latter presents the selectable attribute (Layer Name). The visibility and selectability of layers are configurable in the *Layer Table* pane.

Refer to the **Layer Setting** description for more details.

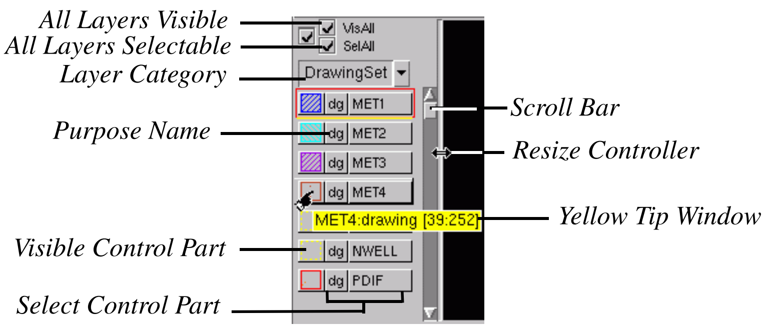


Figure: Layer Table Pane

The category name shown in the **Layer Category** selection list of the *Layer Table* pane is the last selected or deselected category.

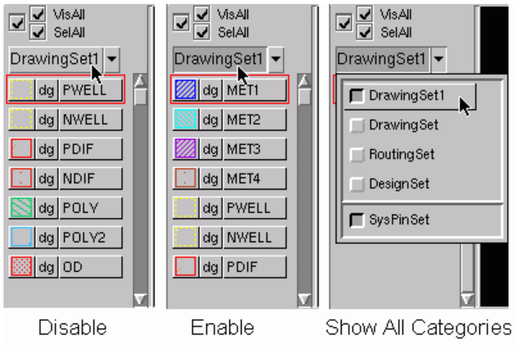


Figure: Layer Category Selection List

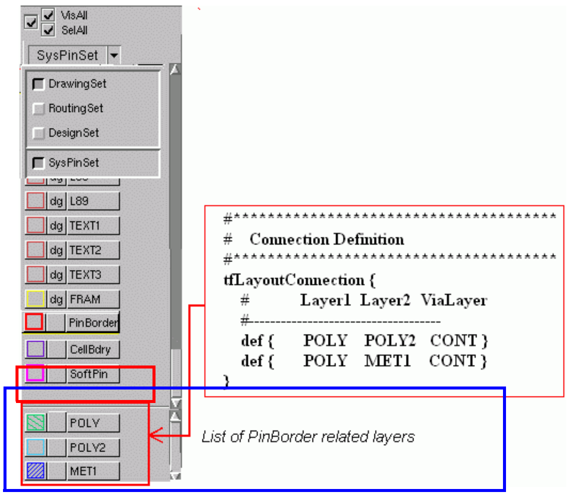
Clicking the name button can select or deselect the category.

Clicking the arrow button shows all categories with the appearance status. The appearance of categories can be configured by checking or unchecking the category button. Refer to the **Category Setting** description for more details.

The built-in **SysPinSet** category in the **Layer Category** selection list collects the system pin layers of the layout in the *Layer Table* pane. This will quickly control the selectability and visibility of related system pin layers in the layout.

The **SysPinSet** category only appears in the *Layer Table* pane of the *Layout* window, it is not configurable in the **Category Setting** tab of the *Layer Table Editor* form.

When the **SysPinSet** category is turned *on*, the PinBorder related layers, which are defined in the technology file section `tfLayoutConnection`, will be listed at the bottom (or right) of the *Layer Table* pane, as shown in the following figure.



```
#***************************************
#   Connection Definition
#***************************************
tfLayoutConnection {
  #       Layer1  Layer2  ViaLayer
  #----------------------------------
  def {    POLY    POLY2   CONT }
  def {    POLY    MET1    CONT }
}
```

Figure: Show List of PinBorder Related Layers when SysPinSet is on

The relationship between the PinBorder layer and the PinBorder related layers in the **SysPinSet** category is explained as follows:

- When the PinBorder layer is set to invisible, all PinBorder related layers will become invisible.
- When the PinBorder layer is set to un-selectable, all PinBorder related layers will become un-selectable.

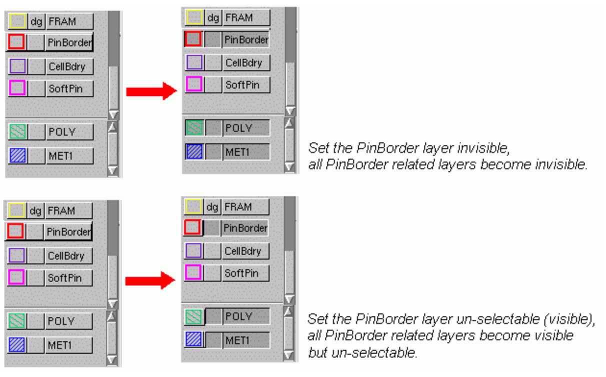
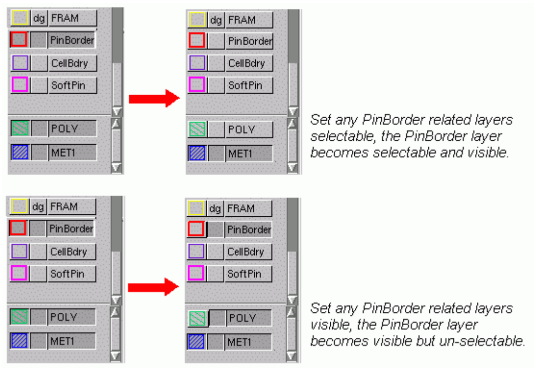

dg FRAM
PinBorder
CellBdry
SoftPin
POLY
MET1
Set the PinBorder layer invisible,
all PinBorder related layers become invisible.
Set the PinBorder layer un-selectable (visible),
all PinBorder related layers become visible
but un-selectable.

Figure: PinBorder Layer and PinBorder Related Layers

- When all PinBorder related layers are invisible, if any of these layers are set to selectable (and visible), the PinBorder layer will become selectable and visible.
- When all PinBorder related layers are invisible, if any of these layers are set to visible, the PinBorder layer will become visible but un-selectable.



Set any PinBorder related layers selectable, the PinBorder layer becomes selectable and visible.

Set any PinBorder related layers visible, the PinBorder layer becomes visible but un-selectable.

Figure: PinBorder Layer and PinBorder Related Layers

Displaying the *Layer Table* pane will make the layout procedures much easier. The *Layer Table* pane can be attached to the bottom, top, left or right of the *Layout* window pane by selecting **Window -> Layer Table -> Attach Bottom**; **Window -> Layer Table -> Attach Top**; **Window -> Layer Table -> Attach Right**; or **Window -> Layer Table -> Attach Left**.

To hide the *Layer Table* pane, choose **Window -> Layer Table -> Hide LayerTbl**.

Scroll Bar

To view other layers in the *Layer Table* pane, use the scroll bar to show other layers.

Resize Controller

Move the mouse cursor over the boundary of the *Layer Table* pane and the mouse cursor becomes a ↕ or ↔ (Resize Controller). To view more or fewer layers, drag the resize controller to change the size of the *Layer Table* pane.

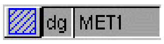
Purpose Name

There are four system purposes defined in the technology file: boundary, net, pin, and drawing. In the *Layer Table* pane, the purpose name is displayed with the first and last characters; for example, the purpose *drawing* is tagged as *dg*.

Refer to the *Customizing Purpose Name* description for more details on the configuration of the length of the purpose name.

Set Layer Selectable/Unselectable

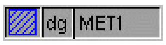
To enable/disable the selectability of a layer, move the mouse cursor over the Select Control Part (Layer Name area), and right-click any layer item. The Select Control Part of the selected layer appears dimmed when it is unselectable, this means that the layer cannot be selected after redrawing the *Layout* window pane. To restore the selectability, right-click the same area again.



Shift-right-click the Select Control Part of a layer item will set the indicated layer selectable, and disable the selectability of other layers. Shift-right-click the same area again will set the layer unselectable, and enable the selectability of other layers.

Set Layer Visible/Invisible

To enable/disable the visibility of a layer, move the mouse cursor over the Visible Control Part (Layer Color/Pattern area), and right-click any layer item. The Visible Control Part of the selected layer appears dimmed when it is invisible, this means that the layer cannot be viewed after redrawing the *Layout* window pane. To restore the visibility, right-click the same area again.



Shift-right-click the Visible Control Part of a layer item will set the indicated layer visible, and disable the visibility of other layers. Shift-right-click the same area again will set the layer invisible, and enable the visibility of other layers.

NOTE: If a layer is set to invisible, it will become unselectable automatically.

Set All Layers Visible

The visibility of all layers can be enabled or disabled by clicking on the **VisAll** button.

Set All Layers Selectable

The selectability of all layers can be enabled or disabled by clicking on the **SelAll** button.

Change Layer Order

Layer order may be changed in the *Layout* window pane when adjusting the drawing order in the layout table pane. The highest drawing priority is at the left/top of the *Layer Table* pane and the lowest is at the right/bottom.

For example, in the following *Layer Table* pane, the first layer shapes will be drawn with the PWELL layer, and then covered with other layer shapes.

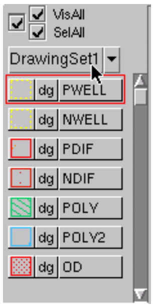


Figure: Layer Table Pane

To change the layer order, select a layer by clicking the left mouse button. Then, move the layer to its destination via a drag-and-drop operation (pressing the middle mouse button). While dragging the item, a yellow line is displayed, indicating the destination position.

Set Layer Color/Pattern Attributes

The *Layer Table Editor* form appears when a layer item is double-clicked. The **User** tab, on the left, is used to modify the layer attributes that are defined in the technology file. Clicking the **Save TF** button saves the modification to the technology file.

In the **User** tab, left-click to select a layer, and modify its color/pattern attributes on the right. The layer name, layer number, and purpose name are editable. When all selections are completed, click the **Modify** button to apply the modifications to the active library. The visibility and selectability settings of a layer in the **User** tab will be synchronized with the *Layer Table* pane.

The **System** tab is used to modify the color/pattern attributes of system features. The layer name, layer number, and purpose name in this **System** tab are not editable. The changes will be saved to a resource file (*laker.rc*), which will use the newly saved settings for the next launch.

Layer Dynamic Selection Field

The layer dynamic selection field at the bottom or rightmost area of the *Layer Table* pane is used to locate specified layers or purposes in the current category set.

In the layer dynamic selection field, enter a string, stream number, or purpose number, and press **Enter** to quickly find the layers matching the filter pattern. The *Layer Table* pane will be refreshed when a filtered pattern is matched. If no layers are matched, the *Layer Table* pane will be empty.

Layer dynamic selection field at the bottom of the Layer Table pane

Layer dynamic selection field at the rightmost area of the Layer Table pane

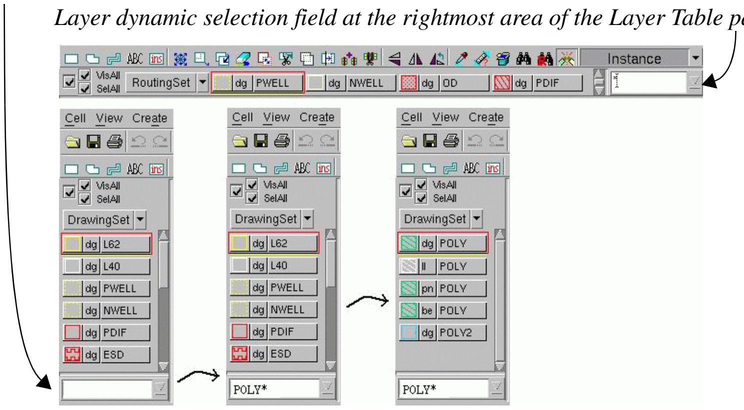


Figure: Layer Dynamic Selection Field

This layer filter saves ten historical filtered patterns, which can be obtained by clicking the arrow button in the dynamic selection field.

The filtered patterns can be specified by number or string. The numerical patterns can be assigned with layer purpose pair or multiple filtered patterns. The string patterns can be assigned with regular expression, layer purpose pair, or multiple filtered patterns.

These filtered patterns apply to the layer selection field of the **Query -> Find**, **Replace**, and **Attribute** commands, and the layer dynamic selection field and active layer selection field in the *Layout* window.

Regular Expressions

The following regular expressions can be used in filtered patterns:

*	Matches any sequence of zero or more characters.
?	Matches any single character. For example, `m?n` will match with `man` and `min`, but not `main`.
[CHARS]	Matches any single character in `CHARS`. If `CHARS` contains a sequence of the form a-b, any character between a and b, inclusive, will match. The characters are case-sensitive. For example, `*[tl]Bdry` will match with `SoftBdry` and `CellBdry`.
\x	Matches the single character x. This avoids special interpretation for any of characters `*?[]\`

Layer Purpose Pair Styles

The colon (`:`) can be used to specify a layer purpose pair with one of the following styles:

- `layerName:purposeName` specifies a layer purpose pair with layer name and purpose name, for example: `MET1:drawing` means that it is a MET1 layer with drawing purpose.
- `layerNumber:purposeNumber` specifies a layer purpose pair with layer number and purpose number, for example: `31:252` means that it is a MET1 with drawing purpose.

- `gdsNumber:dataType` specifies a layer purpose pair with GDS number and data type, for example: `16:0` indicates that it is a MET1 layer with drawing purpose.

Multiple Filtered Patterns

Multiple filtered patterns must be separated by spaces, for example: `POLY* MET? [PN]WELL`.

When specifying multiple filtered patterns, the regular expression and the layer purpose pair can be defined as follows. For example:

- `POLY* MET*:draw*` displays the layers or layer aliases that match with `POLY*`, and also displays the layers that match with `MET*` and the `draw*` purpose.
- `POLY* *WELL:draw* MET*` displays the layers or layer aliases that match with `POLY*` and `MET*`, and also displays the layers that match with `*WELL` and the `draw*` purpose.
- `POLY*:draw* *:p*` displays the layers that match with `POLY*` and the `draw*` purpose, and also displays all layers that match with the `p*` purpose.
- `POLY* 12:*` or `POLY* 12` displays the layers or layer aliases that match with `POLY*`, and also displays the layers that match with layer number `12`.
- `POLY* *:0` or `POLY* :0` displays the layers or layer aliases that match with `POLY*`, and also displays the layers that match with purpose number `0`.
- `POLY* 12:0` displays the layers or layer aliases that match with `POLY*`, and also displays the layers that match with layer number `12` and purpose number `0`.
- `:` displays all layers. This pattern is equal to `*:*`.
- `*` displays all layers. This pattern is equal to `*:*`. When only the symbol `*` is entered in the empty dynamic selection field, the filtered pattern will not be shown and the dynamic selection field will remain empty.

Status Bar

The status bar in the *Design* window consists of the following parts: Tcl Command Line Area, Message Line Area, Active Layer Selection Field, Rule-driven Editing Mode, and Snap Mode.

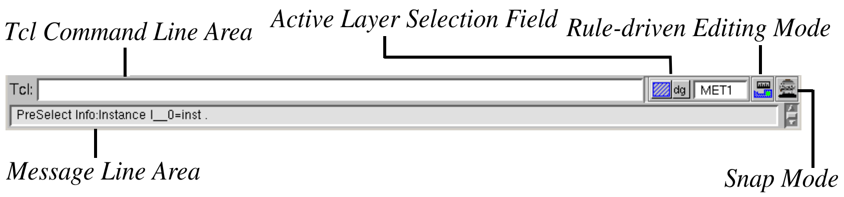


Figure: Status Bar in the Design Window

Tcl Command Line Area

This area executes Tcl commands in the *Layout* window. For example, type the command `lakerZoomAll` and hit the **Enter** key. The current cell view will then fit into the *Layout* window.

Refer to the *Laker Tcl Reference Manual* for more details.

Message Line Area

This area shows the pre-selection information, mode switching messages, and other information, such as the meaning or the brief function of a command. For example, the pre-selection information regarding an active layer, such as *layerName*, *layerNo*, and *PurposeName*, is shown in this field.

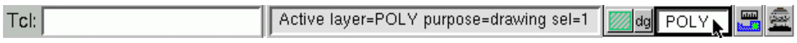
The information format is configurable by using the **Show Information** option in the **Selector** section of the **General** tab in the *Preferences* form (invoked by the **Options -> Preferences** command in the *Main* window), and the number of lines to be displayed in the message line area can be defined in the **Design Window Status Bar** section.

Active Layer Selection Field

This area shows the current drawing layer and the layer information. From this area, the active layer can be selected by using one of the following methods:

- Switch an active layer to another layer by giving a layer name, if the layer exists, in the editable text field, as shown in the following figure.

Specify a layer name in this editable text field



Active layer=POLY purpose=drawing sel=1

Figure: Type Layer Name and Press Enter Key

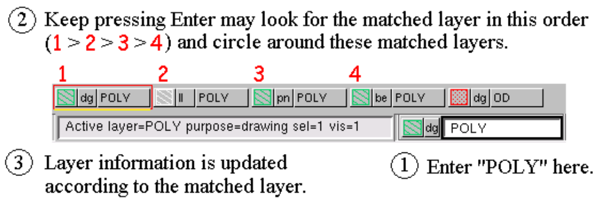
  The layer name and purpose name are case-sensitive. The matched layer is always visible and selectable. Keep pressing the **Enter** key may look for the same matched layer with all its available purposes in turn and circle around these layers if the result is more than one. The layer information is refreshed in the message line area according to the matched layer.

  If the mouse is not anywhere within the editable text field (loosing focus), the application will reset the input layer to active layer.

  All layers, including visible and invisible, can be searched by one of the following actions.

  - Specify full layer name by typing `POLY` in the text field and press **Enter**, the active layer changes to POLY (*drawing*) immediately.

Keep pressing **Enter** may look for the same matched layer with other purpose, as shown in the following figure.


(2) Keep pressing Enter may look for the matched layer in this order (1 > 2 > 3 > 4) and circle around these matched layers.

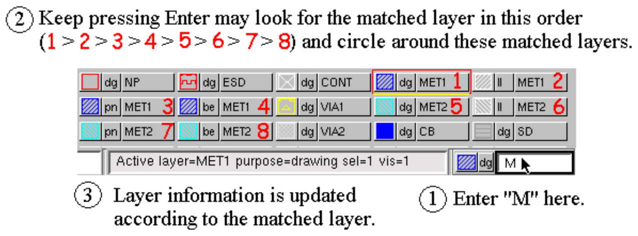
Active layer=POLY purpose=drawing sel=1 vis=1

(3) Layer information is updated according to the matched layer.

(1) Enter "POLY" here.

Figure: Search Matched Layer with All Available Purposes

- Specify the first character or leading characters of layer by typing **`M`** or **`MET`** and press **Enter**, the active layer changes to the first matched layer whose name begins with M or MET, according to the appearance and priority in the layer table, as shown in the following figure.


(2) Keep pressing Enter may look for the matched layer in this order (1 > 2 > 3 > 4 > 5 > 6 > 7 > 8) and circle around these matched layers.

Active layer=MET1 purpose=drawing sel=1 vis=1

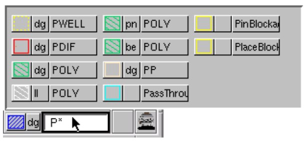
(3) Layer information is updated according to the matched layer.

(1) Enter "M" here.

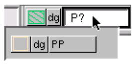
Figure: Search Matched Layer with All Available Purposes

- Search the layer and/or purpose name with regular expressions where the asterisk (*) indicates 0~n characters, and the question mark (?) indicates any characters. For example, typing **`P*`** and press **Enter** may bring up a list of layers whose name begins with P. You may get different results by giving the following combination:

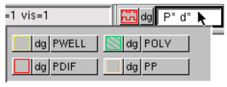
Input	Output
P*	List of layers whose name begins with P.
P?	List of layers whose name begins with P and with another character.
P???	List of layers whose name begins with P and with other three characters.
P* d*	List of layers whose name begins with P and with purpose name begins with d.



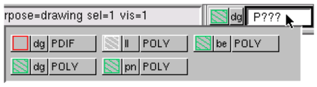
List of layers whose name begins with P.



List of layers whose name begins with P and with another character.



List of layers whose name begins with P and with purpose name begins with d.



List of layers whose name begins with P and with another three characters.

Figure: Search Layer/Purpose with Regular Expression

- Clicking on the purpose name of the active layer appears a list of available layers, as shown in the following figure. The list contents are similar to the *Layer Table* pane. The active layer may be changed by choosing a layer from the list. This function is useful when you hide the *Layer Table* pane, and wish to have a much broader layout space.

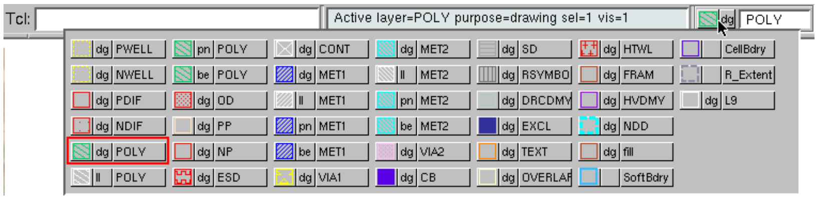


Figure: Left-click the Purpose Name to Show the Layer Table Pane

Status for Rule-driven Editing Mode

This area shows the status for the selected rule-driven editing mode. Using this mode, you can finish your layout task easily. Use **F5** to switch between the modes.

The rule-driven design rules are defined in the technology file section `tfLayoutLayerRule`.

The Rule-driven editing modes are briefly described as follows.

Rule-driven Editing Modes	Description
Display Ruler Mode	A ruler is displayed in the *Layout* window when the Display Ruler Mode is turned *on*.
Rule-driven Mode	With Rule-driven Mode enabled, the object can be calibrated to cope with design rules.
Push-Wire Mode	With Push-Wire Mode enabled, objects can be moved by maintaining the connectivity with correct design rules.

Basic Concept of Rule-driven Editing

Traditional layout editor tools require layout designers to manually measure the distance of two edges, between which there exists a spacing constraint.

While the manual measurement is inaccurate and time-consuming, an innovative layout methodology, rule-driven layout editing, can automatically either *tip* all necessary design rules to inform layout engineers of the occurrence of a violation, or reshape the violation part so that it meets the design rule.

This rule-driven editing capability automatically applies the new design rules after the technology file is updated, allowing designers to operate as usual.

The basic concept and editing modes, including normal, display ruler, rule-driven editing, and push-wire editing, are explained in the following sections.

The rule-driven editing feature incorporates a few display keywords, each of them representing a rule, such as:

w	width rule
s	spacing rule
n	notch rule
o	overlap rule
e	enclosure rule

The concept of the rebounding area is shown in the following figure. When placing the mouse cursor inside the rebounding area, the cursor will be rebounded to the defined rule-distance spot.

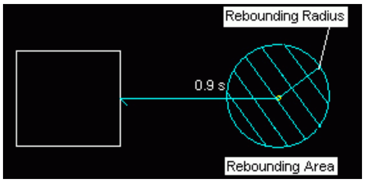


Figure: The Concept of Rebounding Area

Multiple distance-checking rules can be included; however, only the rule that is nearest to the trace cursor will be chosen.

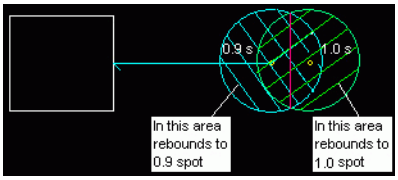

0.9 s
1.0 s
In this area rebounds to 0.9 spot
In this area rebounds to 1.0 spot

Figure: The Concept of Rebounding Area

The rebounding radius and displaying ruler values can be set by turning *on* the **Rule-Driven Threshold** option through the *Preferences* form, invoked by the **Options -> Preferences -> Command** tab under the *Main* window.

Rule-driven Editing Mode

While creating or editing objects, you can change the rule-driven editing mode by using **F5**, as long as the design rules have been defined in the technology file.

Rule-driven editing feature is comprised of Normal mode, Display Ruler mode, Rule-driven mode, and Push-Wire mode. After enabling these modes, pressing the magic key (**Spacebar**) allows you to switch among the modes. When you switch the mode, the current mode will be displayed in the status bar at the same time.

As stretching object with rule-driven and/or display ruler mode, you may enable the **Lock Angles** option, and disable the **Keep Selected Length** option in the *Stretch options* form; otherwise, this may cause the rule-driven and/or display ruler mode to malfunction, and you will see these icons showing in the status bar, and get warning message printed in the message pane of the *Main* window.

Normal Mode

You may work as usual under this normal editing mode (without any design rule iteration guiding you to construct layout).

Display Ruler Mode

The status for Display Ruler mode is represented by . This editing mode notes the distance between the working object and those objects that have the design-rule constraint relationship. The system highlights the violation distance to warn the layout designers whenever the distance violates the design rules, it prompts all necessary design rules as you create or edit an object. For example, when you want to measure the spacing rule between the rectangle being created and an existing object, you can move the mouse cursor from a distance of 1.4 um to 0.8 um, and the ruler will automatically appear and measure the space, as shown in the following figure. In the figure below, 1.4 s means the distance 1.4 um and the spacing rule.

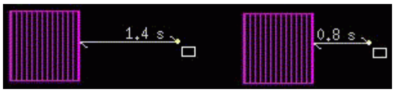


Figure: 1.4 s Means the Distance 1.4 um and the Spacing Rule

To measure the enclosure rule of a rectangle being stretched across an existing object, you can move the mouse cursor from the distance of 0.8 um to 0.5 um, and the ruler will automatically appear and measure the enclosure rule, as shown in the following figure. In the figure below, 0.8 c means the distance 0.8 um and the enclosure rule. In other words, the distance by which the MET1 layer encloses the CONT layer is 0.8 um.

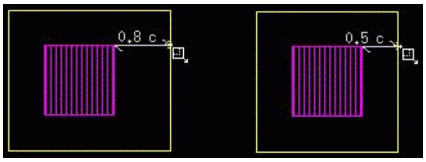


Figure: 0.8 c Means the Distance 0.8 um and the Enclosure Rule

Rule-driven Mode

The status for Rule-driven mode is represented by . This editing mode not only incorporates the functionality of Display Ruler, but also forces the distance between the working object and its neighboring object to fit the minimum rule distance if their proximity violates or approaches the design rule. If there exists more than two constraints for the working objects in question, the minimum rule distance is chosen. It fits the design to the minimum rules if the distances, regarding related objects, approaches the design rule. When creating or editing objects in the rule-driven mode, you will be prompted with the design rules in real-time. In the examples below, we will show you how to create and edit objects on different layers within the rule-driven mode.

Push-Wire Mode

The status for Push-Wire mode is represented by . It allows you to move objects by maintaining the connectivity with correct design rules. Refer to **Edit -> Stretch** for details.

Rule-driven Layout Editing

Enabling the rule-driven editing feature may help you to maintain the accuracy and correctness while editing your design. This section provides a few samples for rule-driven layout editing:

- Creating a Rectangle on the CONT Layer
- Creating a Rectangle on the MET1 Layer
- Copying the Rectangle on the CONT Layer
- Stretching the Rectangle on the MET1 Layer

- Moving Instances, Arrays, or MCells
- Moving Selected Objects

Creating a Rectangle on the CONT Layer

First, we draw a rectangle on the CONT layer. Complete the following steps:

1. Select **Create -> Rectangle**.
2. In the *Layer Table* pane, left-click the CONT layer.
3. Press the magic key (**Spacebar**) to switch the current editing mode to Rule-driven mode.
4. In the *Layout* window pane, left-click to enter the first drawing point for a rectangle, and drag your mouse cursor across the *Layout* window. The outline of a rectangle appears as the mouse cursor moves.
5. The ruler distance appears, as shown in the following figure, only if the drawing area is smaller than the minimum width of the CONT layer, as defined in the `tfLayoutLayerRule` section of the technology file. When the mouse cursor approaches the design rule, the cursor position will automatically incorporate the minimum width defined in the technology file.
6. Left-click again to enter the next drawing point and complete the rectangle. A rectangle has been created on the CONT layer.

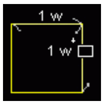


Figure: Ruler Distance Appears If Drawing Area is Smaller Than The Defined Width

Creating a Rectangle on the MET1 Layer

Next, we will draw a rectangle on the MET1 layer, overlapped with the rectangle on the CONT layer. Complete the following steps:

1. In the *Layer Table* pane, left-click the MET1 layer.
2. Left-click in the *Layout* window pane to enter the first drawing point for a rectangle, and drag your mouse cursor to enclose the rectangle on the CONT layer.
3. The ruler distance appears, as shown in the following figure, only if the mouse cursor gets close to the CONT layer. The cursor position will automatically incorporate the minimum enclosure rule, as defined in the technology file.
4. Left-click again to enter the next drawing point and complete the rectangle. A rectangle, overlapped with the rectangle on the CONT layer, has been created on the MET1 layer.

   In this case, the minimum design rule for MET1 to enclose CONT is 0.5um.

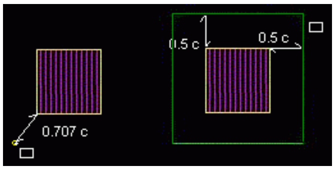


Figure: Ruler Distance Appears If Mouse Cursor Gets Close to The Other Layer

Copying the Rectangle on the CONT Layer

Now, we copy the rectangle you have just created on the CONT layer. Complete the following steps:

1. Select **Edit -> Copy** or use the bind key **c** to invoke the **Copy** command.
2. In the *Layout* window pane, left-click to select the rectangle on the CONT layer. An outline of the selected object appears along with the design rules, as shown in the following figure.

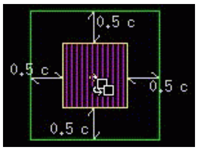


Figure: Object Outline appears along with the Design Rules

3. Move the copied rectangle to the desired place. When the mouse cursor gets close to the existing related rectangle, the copied rectangle will fit the minimum design rule, as shown in the following figure.
4. After you have decided upon a location in which to place the rectangle, left-click to copy it on to the CONT layer.

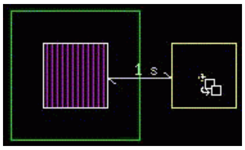


Figure: The Copied Rectangle Will Fit The Minimum Design Rule

Stretching the Rectangle on the MET1 Layer

Next, try to stretch the rectangle created on the MET1 layer toward the rectangle on the CONT layer. Complete the following steps:

1. Select **Edit -> Stretch** or use the bind key **s** to invoke the **Stretch** command.
2. Left-click to select an edge of the rectangle on the MET1 layer. An outline of the selected object appears with the design rules, as shown in the following figure.

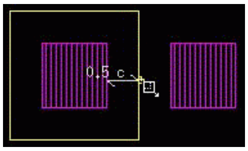


Figure: An Outline of Selected Object Appears with Design Rules

3. Stretch the selected edge to the desired place. When the mouse cursor gets close to the rectangle on the CONT layer, the selected edge will fit the minimum design rule, as shown in the following figure.

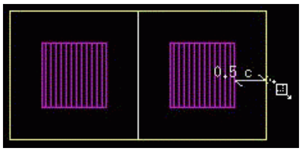


Figure: The Selected Edge Will Fit The Minimum Design Rule

4. Left-click to finish stretching the rectangle. The two rectangles on the CONT layer are now enclosed by the rectangle on the MET1 layer.

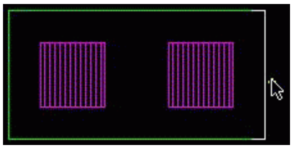


Figure: Rectangles on CONT Layer are Enclosed by the Rectangle on MET1 Layer

Moving Instances, Arrays or MCells

To move instances (including arrays and MCells) with the Rule-driven editing mode, follow the steps below:

1. Use the bind key **m** to invoke the **Move** command.
2. Use the bind key **Shift+f** to set the view level to display all checking shapes of the instance content.
3. In the layout window, left-click to set the reference point and select an instance to move. Only one checking shape that is the nearest to the reference point will be chosen.
4. Move the selected instance to the desired location. When the mouse cursor gets close to the existing related shapes, the checking shape in the selected instance will fit the minimum design rule. The relationship between layers is defined in the Design Rule Description part in technology file.
5. Left-click to place the selected instance on a new location.

Moving Selected Objects

To move objects with the Rule-driven editing mode, follow the steps below:

1. In the layout window, select a few shapes that you want to move.
2. Use the bind key **m** to invoke the **Move** command.
3. Left-click to set the reference point. Only one checking shape that is the nearest to the reference point will be chosen.
4. Move the selected shape to the desired location. When the mouse cursor gets close to the existing related shapes, the checking shape will fit the minimum design rule. The relationship between layers is defined in the Design Rule Description part in technology file.
5. Left-click to place the selected shape on a new location.

Status for Snap Modes

This area shows the status of the selected snap mode. The snap mode controls how the cursor is locked to the grid when creating, editing or selecting objects.

When the snap mode is used in the following commands, it controls how segments or objects snap when creating, moving, re-shaping, selecting or stretching objects: **Align**, **Area Select**, **Bus Route**, **Chop**, **Continue Bus**, **Copy**, **Create Bus**, **Create Dimension Mark**, **Create Guard Ring**, **Create Path**, **Create Pin**, **Create Polygon**, **Create Row**, **Create Ruler Label**, **Create Stack Path**, **Create Text**, **Delete Row**, **Differential Pair Route**, **Measure Distance**, **Measure Point-to-Point Resistance**, **Move**, **ReGuardRing**, **Reshape**, **Rotate**, **Split**, **Split Net**, **Stretch**, **Stretch Capacitor**, **Stretch Contact**, **Stretch Guard Ring**, **Stretch Resistor**, **Stretch Row**, **Symmetric Route**, **Tag Ruler**, and **Yank**.

When creating, editing or selecting segments or objects, the snap mode can be cyclically selected by pressing the **F6** key; or by clicking the **Snap Mode** icon on the status bar, as illustrated and summarized in the following table.

Snap Modes	Description
AnyAngle	Snap segments or objects at any angle.
FixAngle	Snap segments or objects at a user-defined angle. The range of the user-defined angle is from 0 to 180. If the Laker _FPD license is invoked, **FixAngle** can be cyclically selected by **F6** with other snap modes; otherwise it can only be selected from the options form while other snap modes can be cyclically selected by F6.
Orthogonal	Snap segments or objects parallel to the x- or y-axis.
Diagonal	Snap segments or objects parallel to the x- or y-axis, or 45-degree to either axis.

Snap Modes	Description
L90XFirst	Snap two segments forming a 90-degree angle between the points entered, beginning with a horizontal segment.
L90YFirst	Snap two segments forming a 90-degree angle between the points entered, beginning with a vertical segment.
Horizontal	Snap segments or objects parallel to the x-axis.
Vertical	Snap segments or objects parallel to the y-axis.

Selection Schemes

Various selection schemes are provided for different purposes and preferences. The V/E (Vertex/Edge) and OBJ (Object) selection modes are used for objects, and the Device Object Mode is used for MCells. These selection modes are configurable in the *Preferences* form via **Options -> Preferences -> General tab -> Selector -> Mode** and **Device Object Mode** of the *Main* window.

When working with the **Edit** commands, you can customize your own select actions prior to the activation of each command.

Pre-Selection

When you move the trace cursor over an edge, an object or a vertex, a yellow dotted line highlights what you can select. This scheme allows you to preview and choose a target (a vertex, an edge or the whole object) before making any selection.

NOTE: If it is under the [OBJ] selection mode, the object will be highlighted even though the trace cursor is on the edge or vertex.

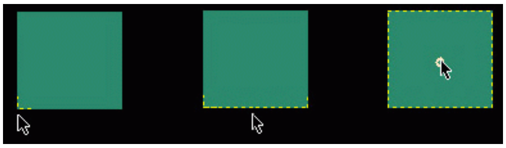


Figure: Pre-select a vertex (left), an edge (middle) or an object (right)

Single Selection

When the edge, object, or vertex is highlighted by the pre-selection scheme, you may left-click or press **Enter** to confirm the selection. The selected object is highlighted with a white solid line and, if any other objects were previously selected, they will be automatically deselected at the moment.

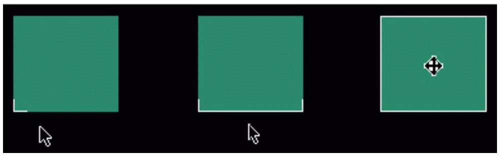


Figure: Select a vertex (left), an edge (middle) or an object (right)

Area Selection

You can select a group of objects by drawing a selection box around them, or by using the available bind key and select options.

To simply select a group of objects, you may drag-left to form a rectangular box over the object(s) you want to select. The object that is entirely inside the area is selected. Under the [V/E] selection mode, if an object is partially inside that area, system will select its edge or vertex that is inside the area. However, under the [OBJ] mode, only the object that is totally inside the area will be selected. If any other objects were selected before this operation, they will be deselected automatically after the action is done.

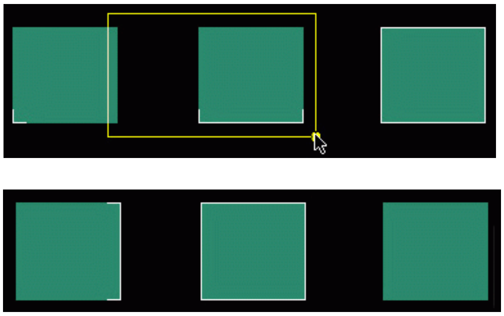


Figure: Drag-left to draw a selection box over a group of objects (top)
Object or edge that is entirely inside the box is selected (bottom)

On the other hand, you may use the bind key **Shift+a** to invoke the **Area Select** command and press **F3** to set your options in the *Area Select options* form, where provides different select options, *Rectangle*, *Polygon* and *Line*, and selection modes, *Enclose* and *Overlap*, to help user to select the desired object(s) efficiently.

Cycle Selection

While making selection on overlapped objects, you can use the **Spacebar** key to pre-select your target edge, vertex or object among the overlaps.

Move your trace cursor over the overlapped area; a yellow dotted line will first highlight the closest edge, vertex or object and show the pre-select information on the status bar underneath. If this is not the target you wish to select, press **Spacebar** to pre-select another that is within the overlapped area. You can repeat this action until you jump to the desired target among the overlaps.

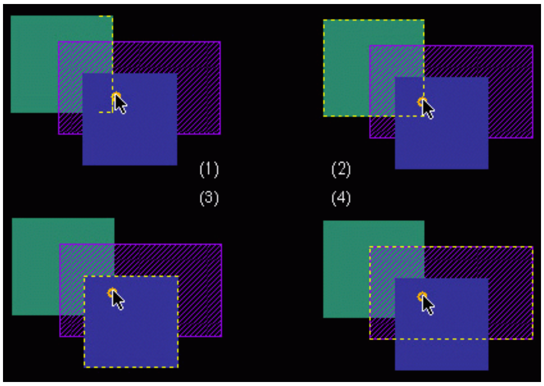


Figure: Press Spacebar to select your target among the overlaps

In this figure, we pre-select an edge of an object first. To see the other available at this pre-selection, press **Spacebar**. You will get the results in (2), (3) and (4). The location of the trace cursor may effect the possible targets.

Multiple Selection

You may make multiple selection when used in combination with the **Shift** key. To select an additional edge, object, or vertex, use Shift-left-click or **Shift+Enter**. To select an additional group of objects, press **Shift** and draw a selection box over the group. Any objects inside the selection box will be selected. The previously selected objects remain selected.

Decrease Selection

You may also deselect an object, among many selected objects, when used in combination with the **Ctrl** key. Use Ctrl-left-click or **Ctrl+Enter** to deselect an additional edge, object, or vertex. To deselect an additional group, press **Ctrl** and draw a selection box over the group. Any objects inside the selection box will be deselected.

To deselect all objects, left-click any empty area in the *Layout* window pane or click the **Deselect All** button on the toolbar.

Used with the Edit Commands

When working with the **Edit** commands (e.g. **Move**, **Stretch**, and **Copy**), you may customize your own selecting ways by using different select actions, as described below, prior to the activation of any editing commands.

When you want to move an object or stretch the edge/vertex of an object, you may select it in two ways:

- **Select the object then invoke the command**: after the operation is done, the object remains selected and the command is terminated. This is known as *One-time Execution*. However, when you copy an object, *only* the newly generated object will be selected after the operation. For example, if you select object A and invoke the **Copy** command, a new object B will then be created and selected; object A will be deselected at this moment.
- **Invoke the command then make the selection**: after the operation is done, the object/edge is deselected; however, you may continue the editing command until you press the **Esc** key to exit. This is known as *Continuable Action*. This continuable command action is available only when the **Command Repeats** option is enabled in the *Preferences* form via **Options -> Preferences -> Command** tab under the *Main* window. If this option is disabled, the executed command working with this selection method will be terminated after the operation. The result is similar to *One-time Execution*.

Quick Start

Overview

Follow the lessons provided in this tutorial, you will gain more practices and skillfully exploit the functionalities of Laker.

This tutorial includes the following lessons:

- *Lesson 1: Import GDSII Stream Design*
- *Lesson 2: Import LEF Design*
- *Lesson 3: Import DEF Design*
- *Lesson 4: Open the Design*
- *Lesson 5: Understand the Design*

The demo cases, which shipped with the Installation CD-ROM, will be used as the examples to guide you through the basic data entry features, editing, and viewing tasks in the Laker layout design system.

The tutorial data resides in the <*install_dir*>*/demo* directory. Copy the demo files from this directory to your working directory.

```
> mkdir <working_dir>
> cp -r <install_dir>/demo <working_dir>
> cd <working_dir>/demo/nECO_Laker/Laker
```

To start the Laker software, execute the *laker* command under UNIX prompt:

```
> laker &
```

Lesson 1: Import GDSII Stream Design

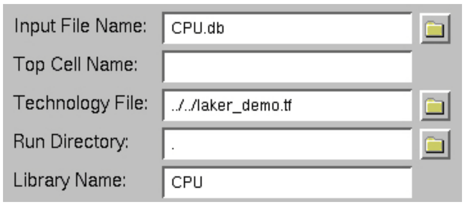
This lesson explains how to import a GDSII stream data.

To import the GDSII stream data into Laker and configure the settings for the importing job, follow the steps below.

1. Invoke **File -> Import -> Stream** in the *Main* window. An *Import Stream* form appears after the command is invoked.
2. In the **Input File Name** text field, specify the stream file, *CPU.db.*

   In the **Technology File** text field, specify the technology file, *<working dir>/ laker_demo.tf*.

   In the **Library Name** text field, specify the library name, *CPU*.

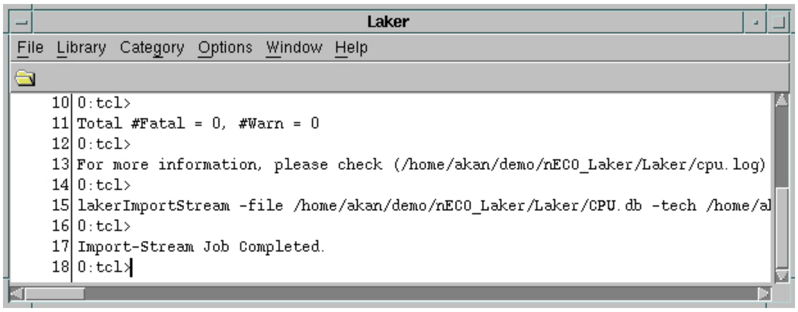


Input File Name:	CPU.db
Top Cell Name:	
Technology File:	../../laker_demo.tf
Run Directory:	.
Library Name:	CPU

Figure: Import Stream File

3. Use the default configuration in the **Basic**, **Advanced**, **Property**, and **Log/Err** tabs.
4. Click **OK** to proceed.

A summary of the data conversion will be displayed in the message pane of the *Main* window, as illustrated in the following figure.



```
0:tcl>
Total #Fatal = 0, #Warn = 0
0:tcl>
For more information, please check (/home/akan/demo/nECO_Laker/Laker/cpu.log)
0:tcl>
lakerImportStream -file /home/akan/demo/nECO_Laker/Laker/CPU.db -tech /home/al
0:tcl>
Import-Stream Job Completed.
0:tcl>
```

Figure: Data Conversion Summary

The detailed information and error report will be recorded in the *gdsIn.log* and *gdsIn.err* files, respectively by default, or they will be saved with the filename specified in the **Log/**

Err File text field of the **Log/Err** tab. These files can be found under the *lakerLog* directory.

Lesson 2: Import LEF Design

This lesson explains how to import an LEF data.

The LEF file includes the basic information to define a leaf cell or a macro cell. This information includes the cell boundary, power/ground pins, signal pins, the origin point of the cell and necessary blockage areas.

Before You Start

Laker supports the DEF/LEF versions 5.3 and 5.4. Please confirm your DEF/LEF format before importing the data into Laker.

Prepare Information File for LEF/DEF

The information file defines the layer name and basic routing information for connection and via layers. The minimum width and space is also included. The following is a sample of the information file.

```
# Define layer mapping relation and information
# layer   Purpose   LEF/DEF     layerType
# Name    Name      layerName   MASTERSLICE/ OVERLAP/CUT
#                               ROUTING  DIRECTION  PITCH  OFFSET WIDTH
#----------------------------------------------------------------------
 CONT     drawing    CONT       CUT
 POLY2    drawing    POLY2      MASTERSLICE
 POLY     drawing    POLY       MASTERSLICE
 PDIF     drawing    PDIF       MASTERSLICE
 MET1     drawing    MET1       ROUTING  HORIZONTAL  1.250  --   0.500
 VIA1     drawing    VIA1       CUT
 MET2     drawing    MET2       ROUTING  VERTICAL    1.400  --   0.600
 VIA2     drawing    VIA2       CUT
 MET3     drawing    MET3       ROUTING  HORIZONTAL  1.300  --   0.600
 VIA3     drawing    VIA3       CUT
 MET4     drawing    MET4       ROUTING  VERTICAL    1.300  --   0.600
 OVERLAP  drawing    OVERLAP     OVERLAP
```

Since initializing the information is a difficult job, Laker provides a utility, `LefIn`, with the `-GenLefInfo` option to parse the layers used in the LEF file, and also generates the information file with the filename giving by the `-info infoFile` option. Refer to the `LefIn` utility in the *Laker Command Reference Manual* for more details.

Since the leaf cell will use as less layer as possible to reduce the wafer processing cost, extra routing layers need to be added manually in the generated information file for DEF file importing.

Import LEF File into Laker

To import the LEF data into Laker and configure the settings for the importing job, follow the steps below.

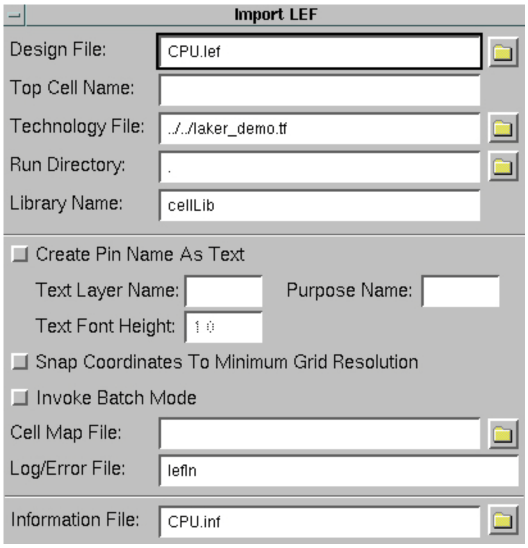
1. Invoke **File -> Import -> LEF** in the *Main* window. An *Import LEF* form appears after the command is invoked.
2. In the **Design File** text field, specify the LEF file, *CPU.lef*.

   In the **Technology File** text field, specify the technology file, *<working dir>/ laker_demo.tf*.

   In the **Library Name** text field, specify the library name, *cellLib*.

   In the **Log/Error File** text field, specify the filename (e.g. *lefIn*) for the log and error files instead of using the default filenames, *LefIn.log* and *LefIn.err*. These files can be found under the *lakerLog* directory.

   In the **Information File** text field, specify the filename (e.g. *CPU.inf*) for the information file.



Design File:	CPU.lef
Top Cell Name:	
Technology File:	../../laker_demo.tf
Run Directory:	.
Library Name:	cellLib

☐ Create Pin Name As Text

Text Layer Name:		Purpose Name:	
Text Font Height:	1.0		

☐ Snap Coordinates To Minimum Grid Resolution

☐ Invoke Batch Mode

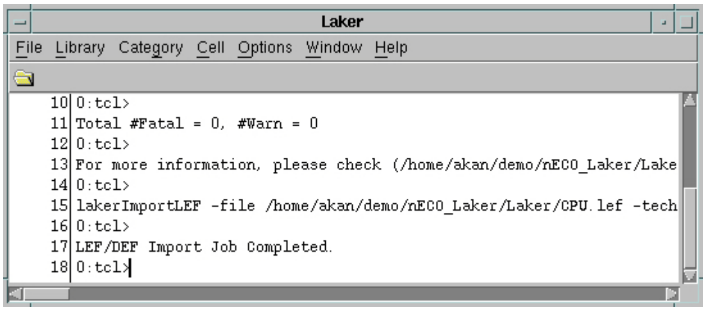
Cell Map File:	
Log/Error File:	lefIn
Information File:	CPU.inf

Figure: Import LEF File

3. Use other default settings in the *Import LEF* form, and click **OK** to proceed.

   All of the cells in the LEF file will be translated. These cells will be stored as the *abstract* views instead of the *layout* views. The abstract view data does not include a complete design information and it cannot be exported as the stream data for tape out purposes.

A summary of the data conversion will be displayed in the message pane of the *Main* window, as illustrated in the following figure.



```
0:tcl>
Total #Fatal = 0, #Warn = 0
0:tcl>
For more information, please check (/home/akan/demo/nECO_Laker/Lake
0:tcl>
lakerImportLEF -file /home/akan/demo/nECO_Laker/Laker/CPU.lef -tech
0:tcl>
LEF/DEF Import Job Completed.
0:tcl>
```

Figure: Data Conversion Summary

Lesson 3: Import DEF Design

This lesson explains how to import a DEF data.

To import the DEF data into Laker and configure the settings for the importing job, follow the steps below.

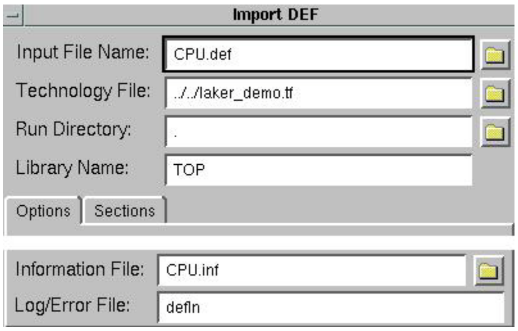
1. Invoke **File -> Import -> DEF** in the *Main* window. An *Import DEF* form appears after the command is invoked.
2. In the **Input File Name** text field, specify the DEF file, *CPU.def*.

   In the **Technology File** text field, specify the technology file, *<working dir>/ laker_demo.tf*.

   In the **Library Name** text field, specify the library name, *TOP*.

   In the **Log/Error File** text field, specify the filename (e.g. *defIn*) for the log and error files instead of using the default filenames, *DefIn.log* and *DefIn.err*. These files can be found under the *lakerLog* directory.

   In the **Information File** text field, specify the filename (e.g. *CPU.inf*) for the information file.



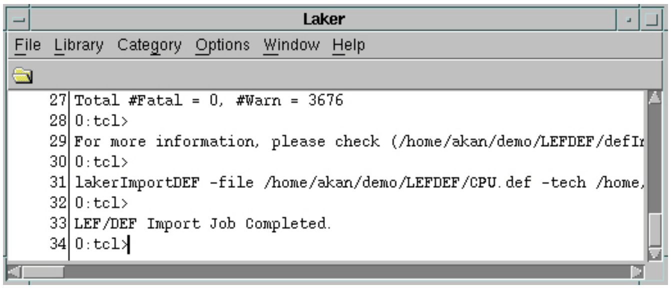
Input File Name:	CPU.def
Technology File:	../../laker_demo.tf
Run Directory:	.
Library Name:	TOP
Information File:	CPU.inf
Log/Error File:	defIn

Figure: Import DEF File

3. Use other default settings in the *Import DEF* form, and click **OK** to proceed.

   During the import process, all valid Laker libraries will be searched to find the master cell name with *abstract* views. If the *abstract* views cannot be found in the libraries, the *layout* views will be the second priority. If both the *abstract* and *layout* views are not found in the libraries, the master cells will be treated as undefined cells.

A summary of the data conversion will be displayed in the message pane of the *Main* window, as illustrated in the following figure.



```
Total #Fatal = 0, #Warn = 3676
0:tcl>
For more information, please check (/home/akan/demo/LEFDEF/defIr
0:tcl>
lakerImportDEF -file /home/akan/demo/LEFDEF/CPU.def -tech /home,
0:tcl>
LEF/DEF Import Job Completed.
0:tcl>
```

Figure: Data Conversion Summary

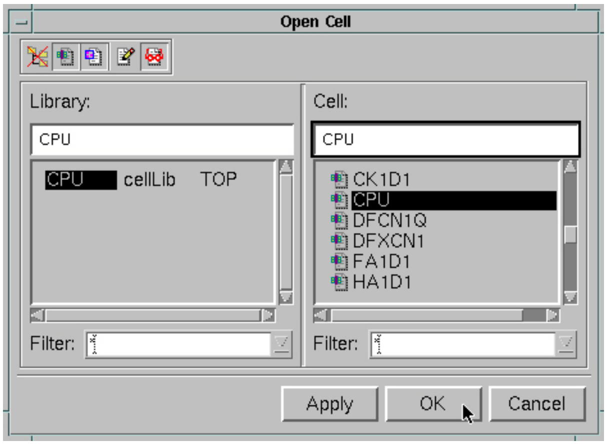
Lesson 4: Open the Design

If you have completed Lesson 1 to Lesson 3, you created three Laker libraries: *CPU*, *cellLib* and *TOP*. In this lesson, we will try to open different designs from the *CPU* and *TOP* libraries.

Open a Stream Design

Follow the steps below to open the design imported from the GDSII stream file.

1. Invoke **File -> Open** in the *Main* window. An *Open Cell* form appears after the command is invoked.
2. In the *Open Cell* form, select the *CPU* library and the *CPU* cell, and click **OK**.


Library: CPU

Cell: CPU

Figure: Select CPU Cell from CPU Library

3. The *CPU* cell is then loaded in the *Layout* window. With the Laker Viewer license, the opening cell is always set to **Read Only** mode. You may find that most of the editing toolbar buttons are not available in the *Layout* window.

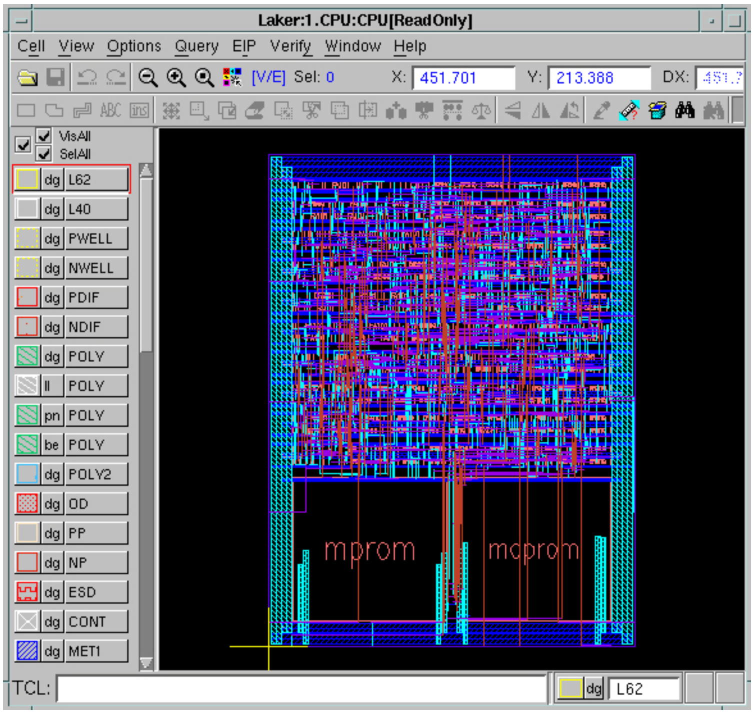


Figure: CPU Cell is Loaded in Layout Window

4. The initial loaded layout is displayed at the minimum view level (level=0). Let's try to use the zooming feature to get a close-up view of the specific area and see the view level.

   First, use the bind key **z** to invoke the **Area Zoom** command. Next, left-click to insert the first point, and then move your mouse cursor. Finally, left-click to insert the last point to finish drawing a rectangular box on a layout area to enlarge the view.

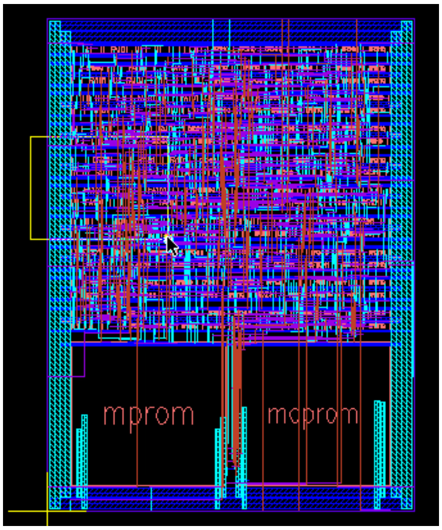


Figure: Drag-left to Invoke Area Zoom Command

5. The zoomed area is displayed at the minimum view level, as illustrated in the following figure.

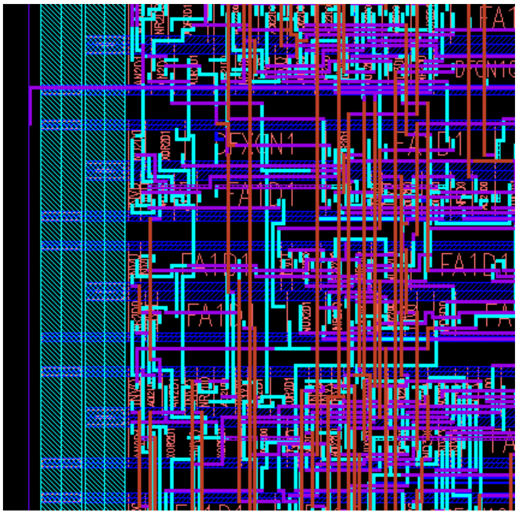


Figure: Layout at Minimum View Level

6. Press **Shift+f** to change the view level to the maximum level (level=49). You will see every detail of the layout design in the *Layout* window pane.

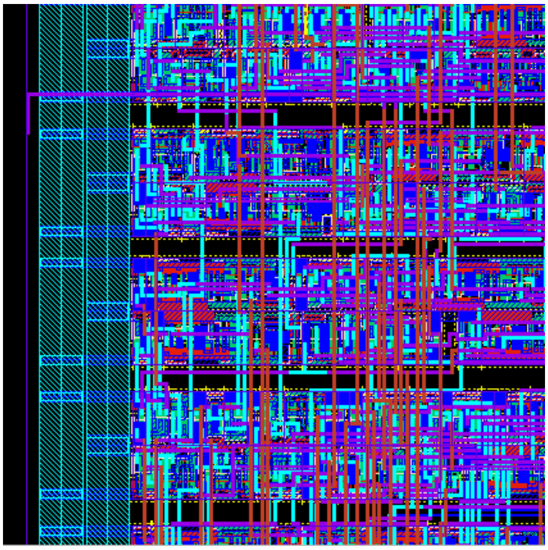


Figure: Layout at Maximum View Level

7. To close the *Layout* window, select **Cell -> Close**.

Open DEF Design

Let’s open the design imported from the DEF file and see the difference between Import Stream and Import DEF. Follow the steps below to open the design imported from the DEF file.

1. Invoke **File -> Open** in the *Main* window. An *Open Cell* form appears after the command is invoked.
2. In the *Open Cell* form, select the *TOP* library and the *CPU* cell, and click **OK**.

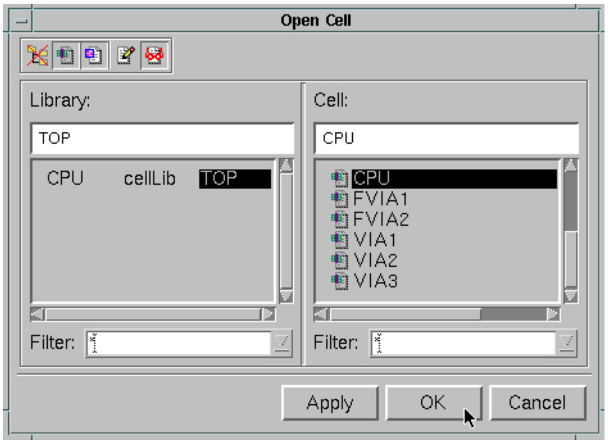


Figure: Select CPU Cell from TOP Library

3. The CPU cell is then loaded in the *Layout* window.

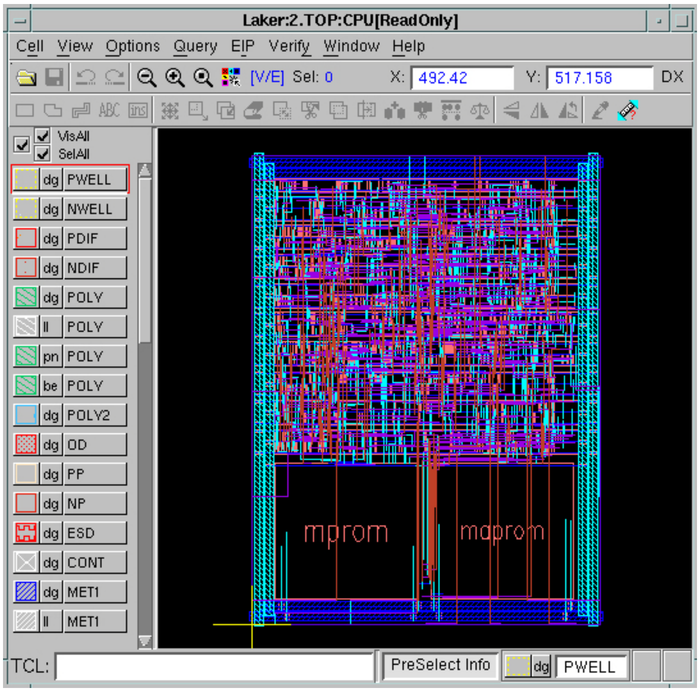


Figure: CPU Cell is Loaded in Layout Window

4. The layout is displayed at the minimum view level (level=0). Let’s try to use the **Area Zoom** feature to get a close-up view of the specific area and see the view level.

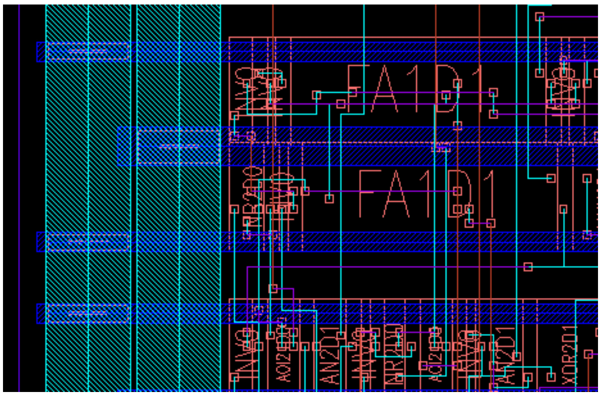


Figure: DEF Design with Minimum View Level

5. In this case, we map the master cell to the LEF-In data, we just see the cell abstract data, as illustrated in the following, instead of the complete layout data.

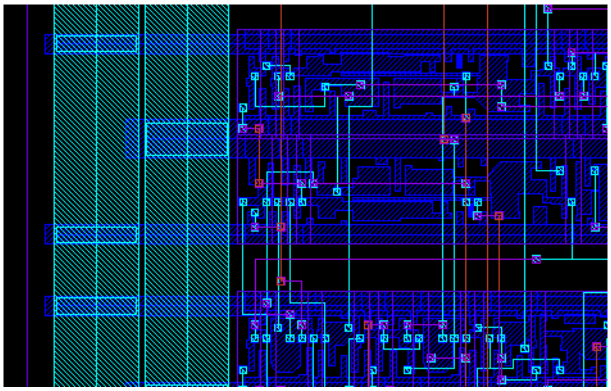


Figure: DEF Design with Abstract Cell Data from LEF File

If you import the stream file and LEF file of all leaf cells, you can select **View -> View Instance** to switch the display view name between the *layout* and *abstract* views.

6. To close the *Layout* window, select **Cell -> Close**.

Lesson 5: Understand the Design

This lesson explains how to use the **Highlight Net** and **Hierarchical Net Tracer** functions to understand the layout data.

Highlight Net

Layout designers can invoke **Query -> Highlight Net** in the *Layout* window to get the *Highlight Net* form.

All net names in the opened cell are extracted from Laker database and listed in the **Net Name** list pane.

You can directly enter the desired net name in the **Net Name** text field and click the **Apply** button to see the whole routing topology.

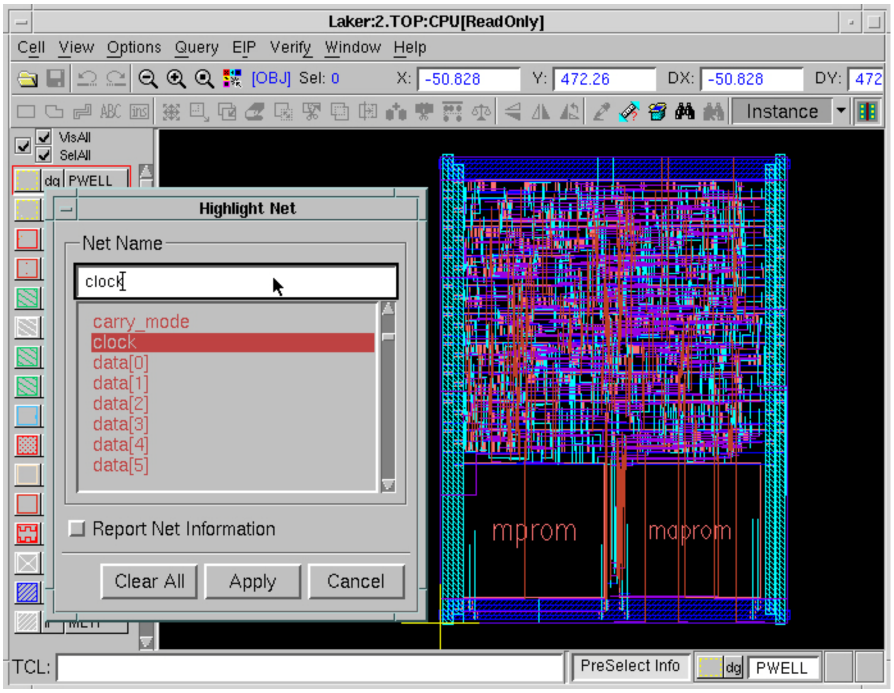


Figure: Enter Desired Net Name in the Highlight Net Form

To query the net name, simply move your mouse cursor onto the net in the layout. The corresponding net name will be shown in a yellow tip window.

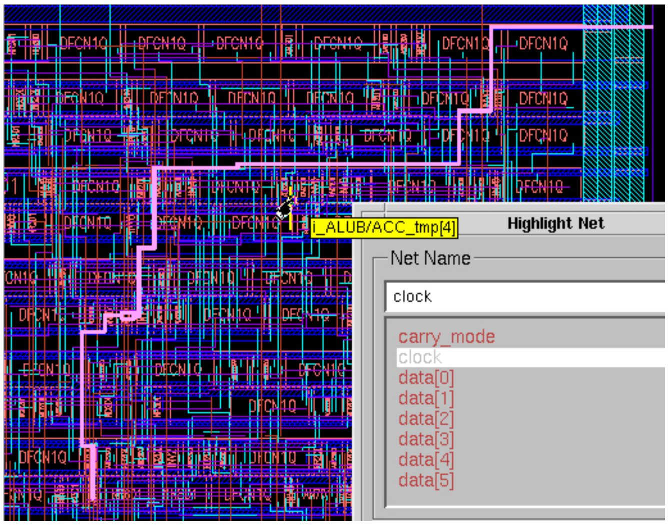
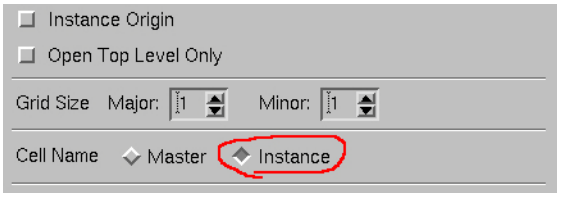


Figure: Highlight Net Result with Tip for Net Name Reporting

To see the hierarchical instance name instead of the master cell name, change the display setting of the cell name by choosing **Instance Cell Name** in the **Options -> Preference -> Display** tab of the *Main* window.



☐ Instance Origin

☐ Open Top Level Only

Grid Size Major: 1 Minor: 1

Cell Name Master [✓ Instance]

Figure: Preference Setting for Display of Cell Name

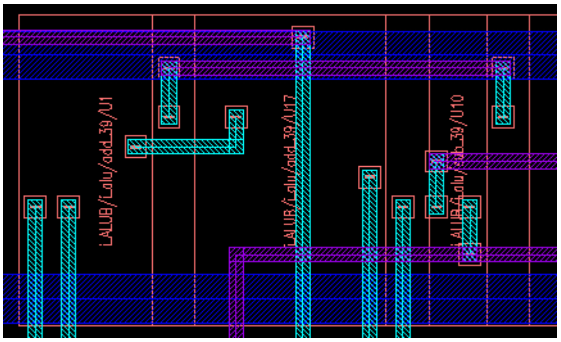
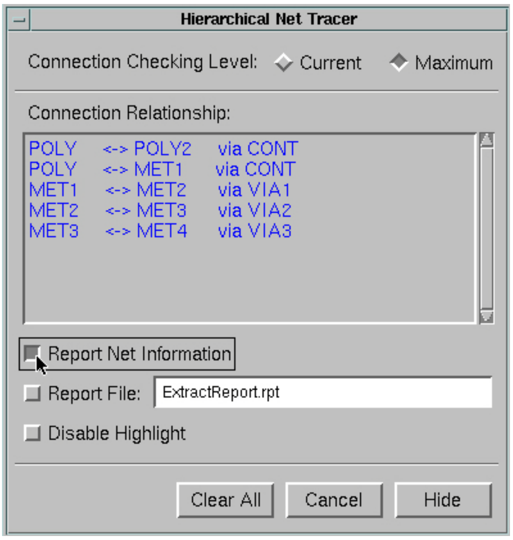


Figure: Show Instance Name on Layout Pane

Hierarchical Net Tracer

Before utilizing this hierarchical net tracer, the `tfLayoutConnection` section must be defined in the technology file that you use during the import of the stream or LEF/DEF designs.

Layout designers can invoke **Query -> Hierarchical Net Tracer** to invoke the Hierarchical Net Tracer function. After the command is invoked, press **F3** to get the *Hierarchical Net Tracer* form.


Hierarchical Net Tracer

Connection Checking Level: Current [✓ Maximum]

Connection Relationship:

```
POLY   <-> POLY2   via CONT
POLY   <-> MET1    via CONT
MET1   <-> MET2    via VIA1
MET2   <-> MET3    via VIA2
MET3   <-> MET4    via VIA3
```

[✓ Report Net Information]

Report File: ExtractReport.rpt

Disable Highlight

Figure: Hierarchical Net Tracer Form

Click a desired polygon shape in the layout to perform the tracing job. Ten different colors are cyclic used to highlight the traced results.

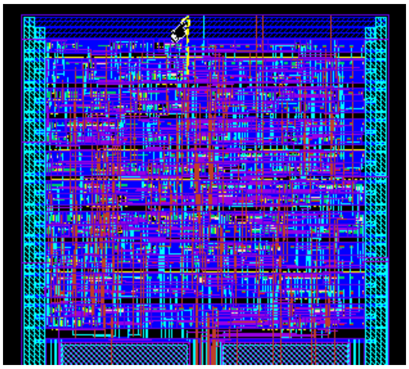
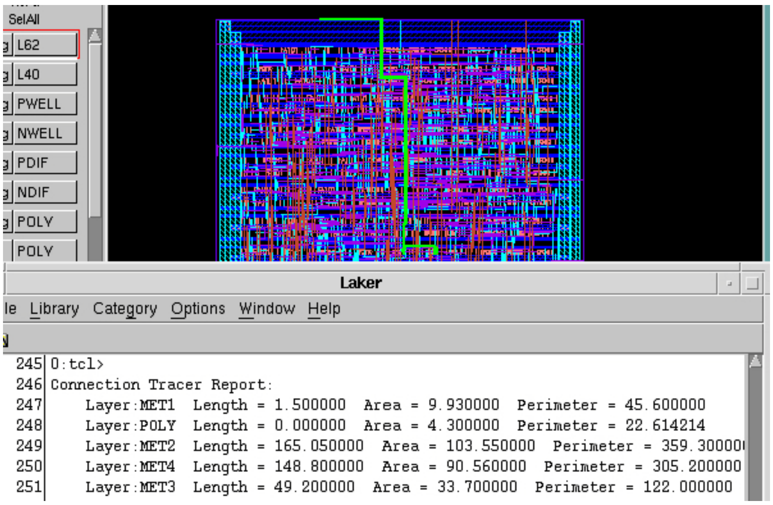


Figure: Click a Polygon Shape and Trace the Net

The net information will be displayed in the message pane of the *Main* window.



```
245 0:tcl>
246 Connection Tracer Report:
247     Layer:MET1  Length = 1.500000   Area = 9.930000   Perimeter = 45.600000
248     Layer:POLY  Length = 0.000000   Area = 4.300000   Perimeter = 22.614214
249     Layer:MET2  Length = 165.050000  Area = 103.550000  Perimeter = 359.30000
250     Layer:MET4  Length = 148.800000  Area = 90.560000  Perimeter = 305.200000
251     Layer:MET3  Length = 49.200000  Area = 33.700000  Perimeter = 122.000000
```

Figure: Tracing Report in the Message Pane

If the hierarchical net tracer cannot finish the tracing job, Laker will show a progress bar on the screen and allow the user to click the **STOP** button to terminate the tracing job.

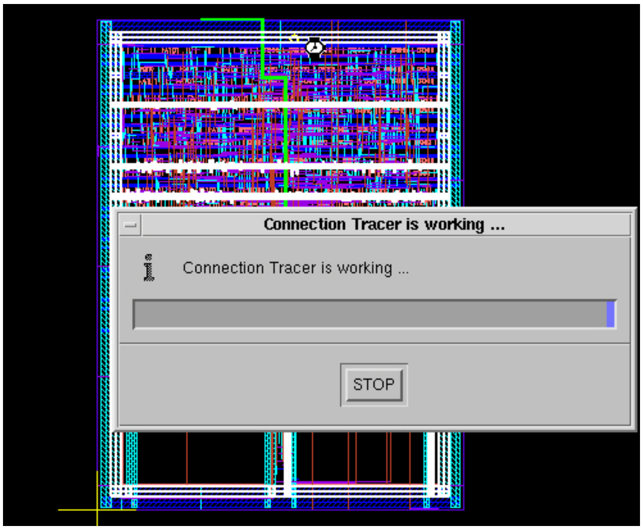

Connection Tracer is working ...
Connection Tracer is working ...
STOP

Figure: Click STOP to Terminate Tracing Job

Laker does not provide the incremental hierarchical net tracing function. The terminated results cannot be reused for the next tracing.

UDD Tutorial

Before You Start

Create a New UDD in the Main Window

1. Invoke **Cell -> UDD -> New UDD** from the *Main* window to start creating a new UDD.
2. Choose a destination library, and then type a new device name in the **Cell Name** text field.
3. Click **OK** to confirm the creation of new UDD, which is created with *device* view.
4. Open this new UDD by selecting **File -> Open** from the *Main* window.
5. Go to the relative library and open the UDD by double-clicking on it.

Open the UDD in the Design Window

In the *Design* window, you may start designating the parameters to be applied to the shapes in your design by using the toolbar icons or commands from UDD pull-down menu. But, because the aim of this chapter is not to explain the details of them, only brief description is introduced here.

Toolbar Icons

A set of frequently used commands is collected in the toolbar. Most of them are for defining procedure statements, and three of them (**Rule Finder**, **Preview**, and **Debug**) are accessories.

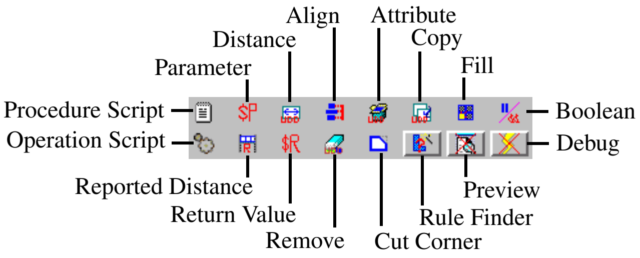


Figure: Toolbar Icons for UDD Creation

Procedures are categorized into:

- **Default Procedure**: Each default procedure is evaluated independently.
- **Sequential Procedure**: The order of sequential procedures regulates the final result. The **Copy**, **Fill**, **Boolean** and **Remove** procedures belong to this type.

UDD Pull-down Menu

This menu, as shown in the following figure, consists of a collection of non-frequently used commands.

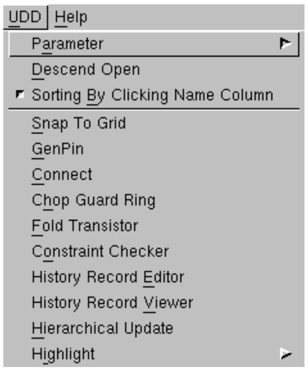


Figure: UDD Pull-down Menu

Creating a User-defined Device

This section shows you how to create user-defined devices (UDDs) with parameterized procedures, which contain procedure statements to control the size, shape, or contents of a cell instance.

In this tutorial, you will be

- Importing UDD Layout
- Creating Shapes and Evaluating Constraints
- Correcting the UDD
- Saving the UDD

Importing UDD Layout

An UDD can be constructed by these valid objects:

- Manhattan shapes, e.g. rectangles and polygons.
- Instances, instance arrays, Tcl PCells, and UDDs.

- Text labels.

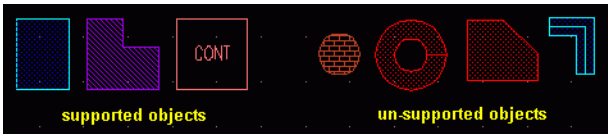


Figure: Supported and Un-supported Objects

You may re-use existing UDD layout by importing UDD procedure script with the **Procedure Script** button on the toolbar. Then, evaluate the constraints to fit your design needs.

Alternatively, you may create a brand new UDD layout with valid objects and constraints.

Creating Shapes and Evaluating Constraints

In this tutorial, we will create a transistor-like UDD, as shown in the following figure.

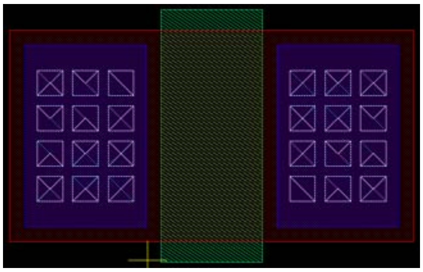


Figure: Transistor-like UDD

To do so, follow the steps below.

1. In the layout window, draw a contact device and a few simple rectangles to form the elements of transistor, as shown in the following figure.

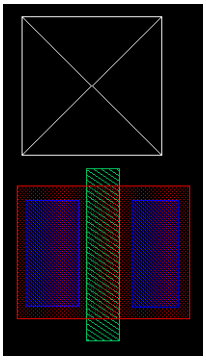
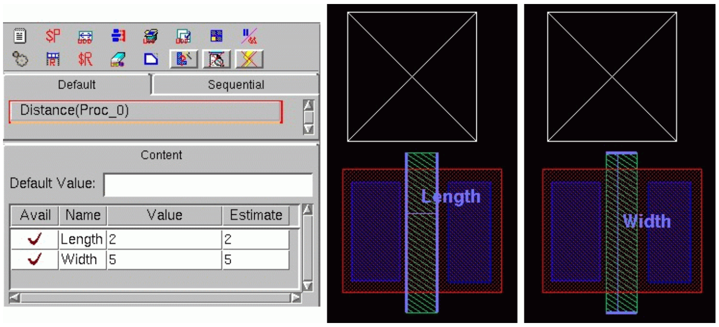


Figure: Create Necessary Objects for UDD

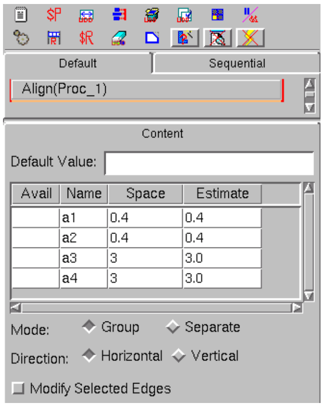
2. Next, assign parameterized constraints to the shapes. In this tutorial, we'll define the procedures by this order: Distance > Align > Fill > Align > Align > Remove.
3. Click the **Distance** button to insert a Distance procedure. The length of transistor gate is defined as 2 and the width is 5, as shown in the following figure.



Avail	Name	Value	Estimate
✓	Length	2	2
✓	Width	5	5

Figure: Distance Procedure

4. Click the **Align** button to add the first Align procedure, and define four align constraints, `a1`, `a2`, `a3`, and `a4`, as follows.



Align(Proc_1)

Default Value:

Avail	Name	Space	Estimate
	a1	0.4	0.4
	a2	0.4	0.4
	a3	3	3.0
	a4	3	3.0

Mode: Group Separate

Direction: Horizontal Vertical

Modify Selected Edges

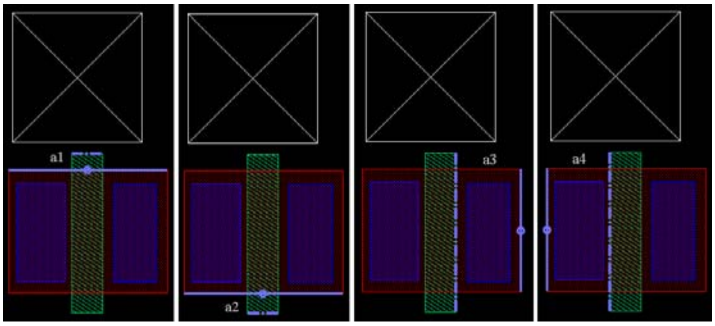
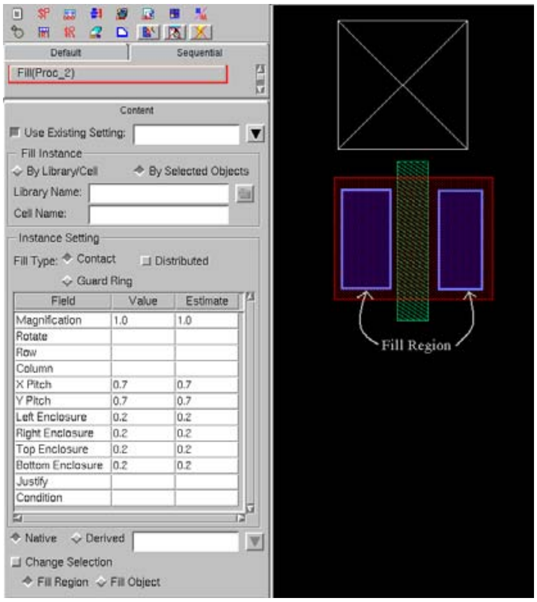


Figure: Align Procedure

5. Click the **Fill** button to insert a Fill procedure. You may select a fill region for insertion, and also define the fill constraints in Instance Setting group box.



Use Existing Setting:

Fill Instance

By Library/Cell [✓ By Selected Objects]

Library Name:

Cell Name:

Instance Setting

Fill Type: [✓ Contact] Distributed Guard Ring

Field	Value	Estimate
Magnification	1.0	1.0
Rotate		
Row		
Column		
X Pitch	0.7	0.7
Y Pitch	0.7	0.7
Left Enclosure	0.2	0.2
Right Enclosure	0.2	0.2
Top Enclosure	0.2	0.2
Bottom Enclosure	0.2	0.2
Justify		
Condition		

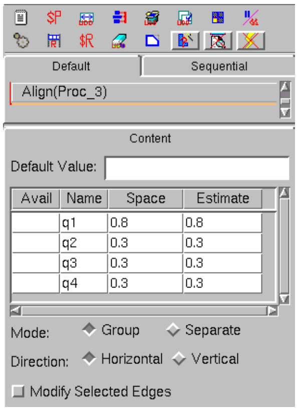
[✓ Native] Derived

Change Selection

[✓ Fill Region] Fill Object

Figure: Fill Procedure

6. Click the **Align** button to add the second Align procedure, and define four align constraints, `q1`, `q2`, `q3`, and `q4`, as follows.



Default Value:

Avail	Name	Space	Estimate
	q1	0.8	0.8
	q2	0.3	0.3
	q3	0.3	0.3
	q4	0.3	0.3

Mode: [✓ Group] Separate

Direction: [✓ Horizontal] Vertical

Modify Selected Edges

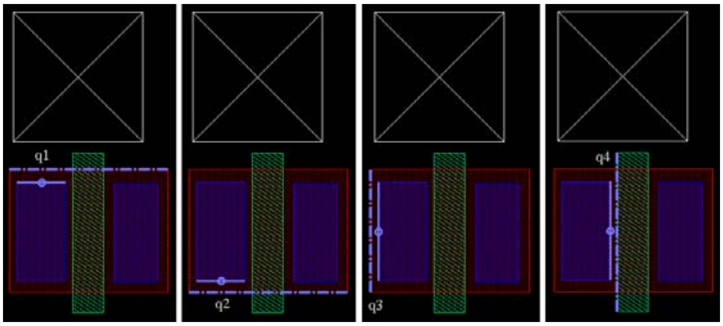
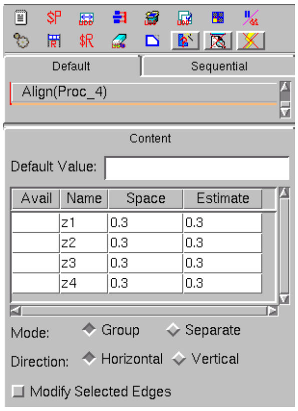


Figure: Align Procedure

7. Click the **Align** button again to add the third Align procedure, and define four align constraints, `z1`, `z2`, `z3`, and `z4`, as follows.



Default Value:

Avail	Name	Space	Estimate
	z1	0.3	0.3
	z2	0.3	0.3
	z3	0.3	0.3
	z4	0.3	0.3

Mode: ◆ Group ◇ Separate

Direction: ◆ Horizontal ◇ Vertical

☐ Modify Selected Edges

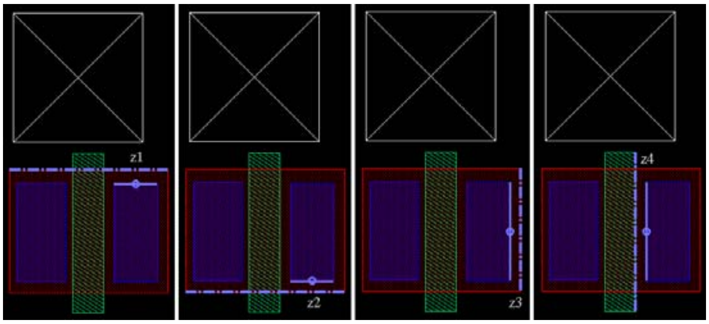


Figure: Align Procedure

8. Click the **Remove** button to remove the unnecessary contact shape.

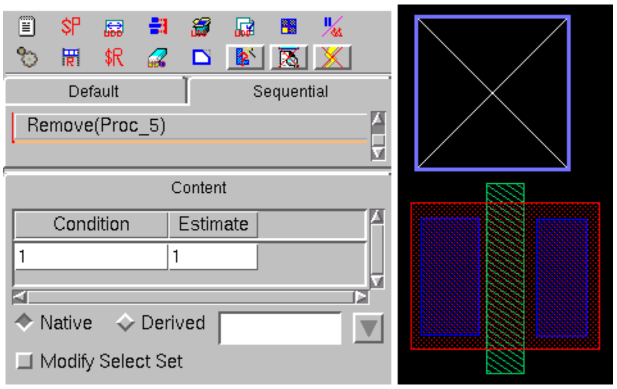


Figure: Remove Procedure

Now, we have finished defining all design constraints.

Correcting the UDD

Let's check if these design constraints meet our requirements. To do so,

1. Click the **Preview** button and you'll see the following result.

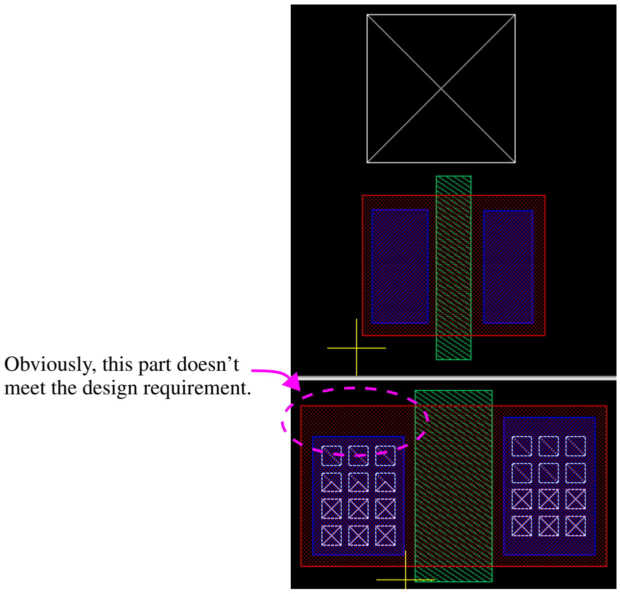


Figure: Preview UDD

2. Obviously, the left part of the transistor doesn't not meet our needs. To fine-tune this part, we have to find out which procedure it belongs to.

3. Select the topmost edge of metal area in the layout window and then click the **Rule Finder** button to find out the constraint. The *Rule Finder* form appears and the associated preview result will also be highlighted if the **Highlight Object on Preview Area** option is turned on.

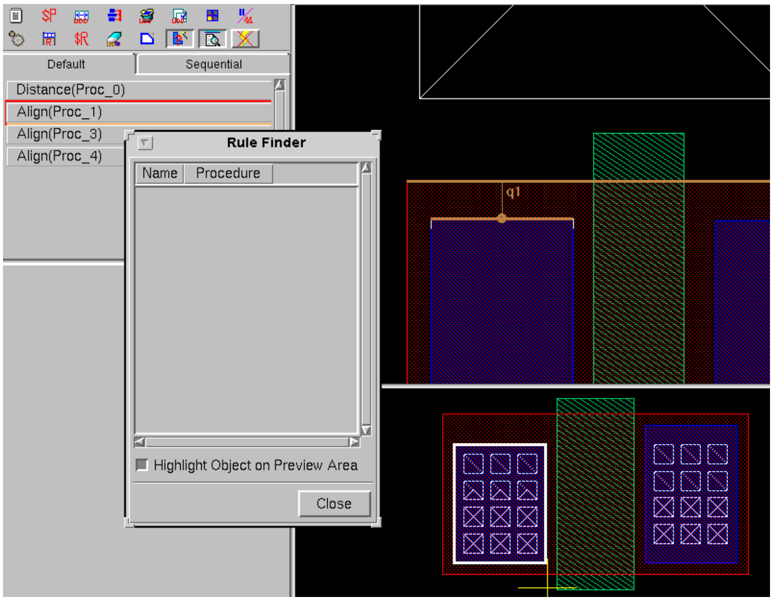


Figure: Rule Finder

4. Next, click-left on the `q1` constraint. Its belonging procedure will appear in the *Rule Finder* form. Double-click `Proc_3`, it will bring you to `q1`'s setting box, so that you can quickly locate the constraint `q1` and all its associated information, as illustrated in the dotted circles below.

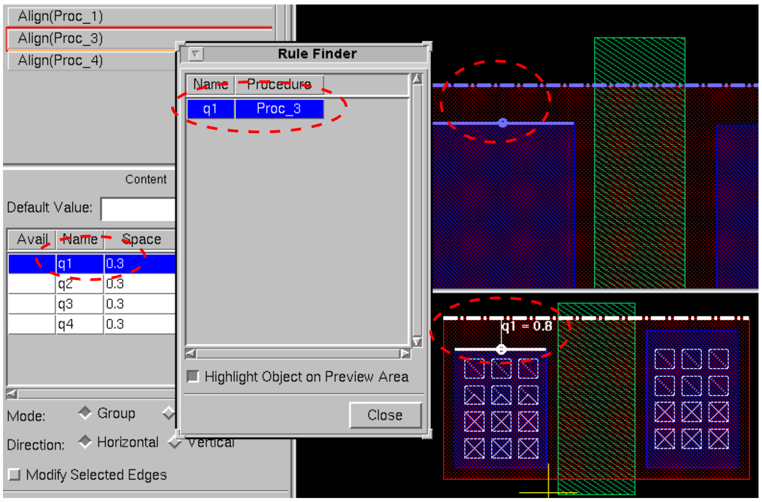


Figure: q1 and all its associated information

5. Change `q1`'s spacing from `0.8` to `0.3`, and then hit **Enter**.
6. Close the *Rule Finder* form, and click the **Preview** button again.
7. The result is illustrated as follows.

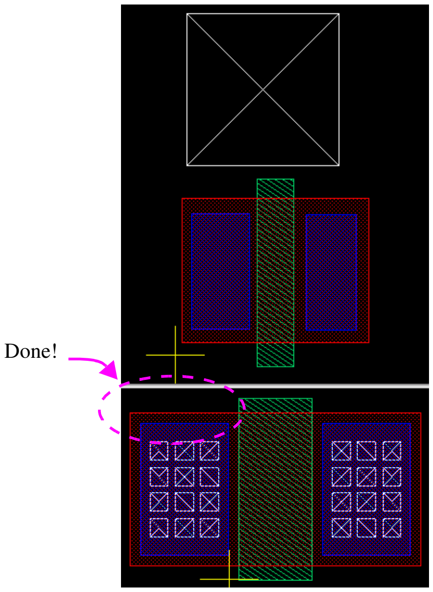


Figure: Preview Result after Correction

Saving the UDD

1. To save the UDD, select **Cell -> Save**.
2. Choose **Cell -> Close** to close the design.

Summary

In this tutorial, you have learned how to create and modify UDDs. Specifically, you

- Created shapes and evaluated constraints.
- Corrected the UDD.
- Saved the UDD.

Spare Cell Management

Overview

This lesson explains how to use the Spare Cell function to collect the spare cell information in the stream or LEF/DEF design and also explains how to find the space cells in the *Layout* window.

How to Use It

This lesson will use the stream design as the example. Let's open the *CPU* cell, that you have used in the *Quick Start* chapter, from the *CPU* library.

Invoke **Query -> Spare Cell** to open the *Spare Cell* form.

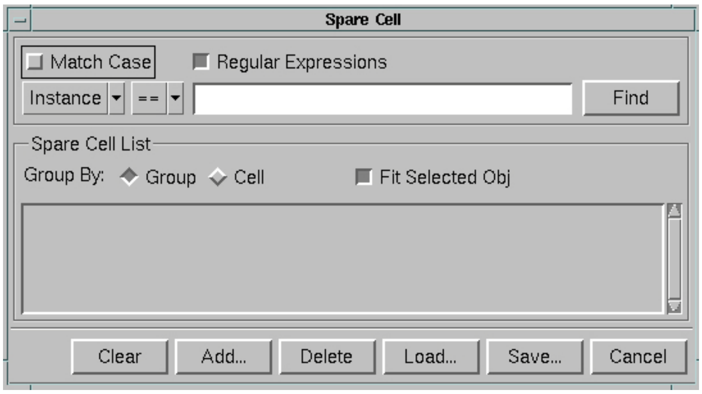
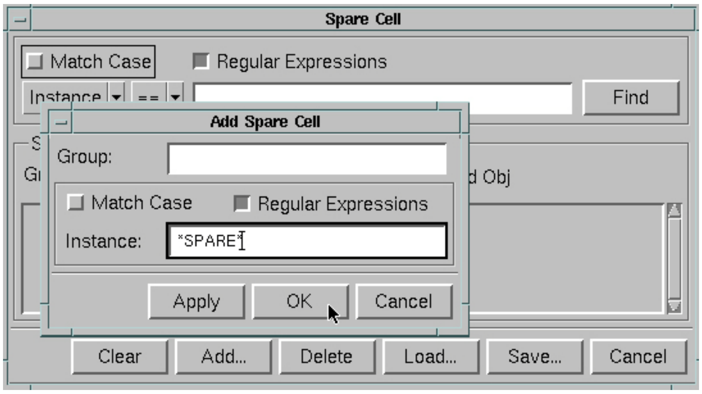


Figure: Spare Cell Form

If you have the list of pre-defined spare cells, click the **Load** button and import into this form.

You may click **Add** button to add the spare cell in the form.



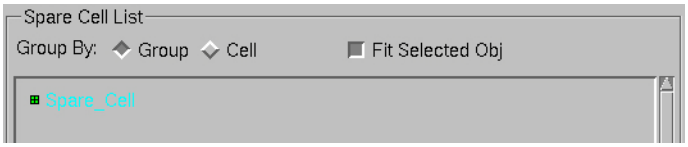
Group:

Match Case [✓ Regular Expressions]

Instance: *SPARE*

Figure: Add Spare Cell Form

In this case, the instance names for these spare cells own the keyword, *SPARE*. Let's use the regular expression style to search in the Laker database. If you don't specify the group name in the **Group** text field, Laker will automatically create the group with the default name, *Spare_Cell*.

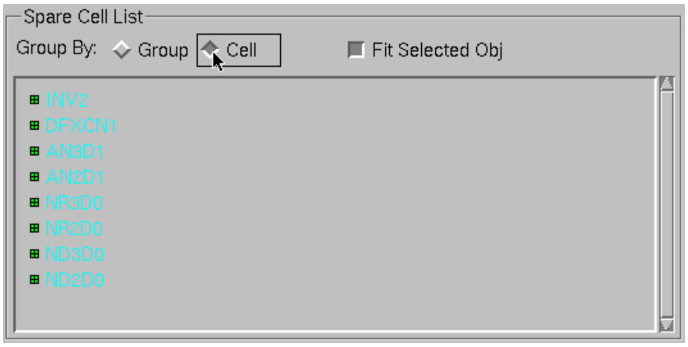


Group By: [✓ Group] Cell [✓ Fit Selected Obj]

Spare_Cell

Figure: Spare Cell List with Search Result

Here, you can choose to show the list by **Group** or by **Cell**. Let's list the spare cells by **Cell**. Click the **Cell** button to change the **Group By** option.



Group By: Group [✓ Cell] [✓ Fit Selected Obj]

INV2
DFXCN1
AN3D1
AN2D1
NR3D0
NR2D0
ND3D0
ND2D0

Figure: Group By Cell

To highlight the spare cells in the layout pane, simply click the **Cell** group to highlight the whole spare cell group or click a single spare cell to highlight the single spare cell.

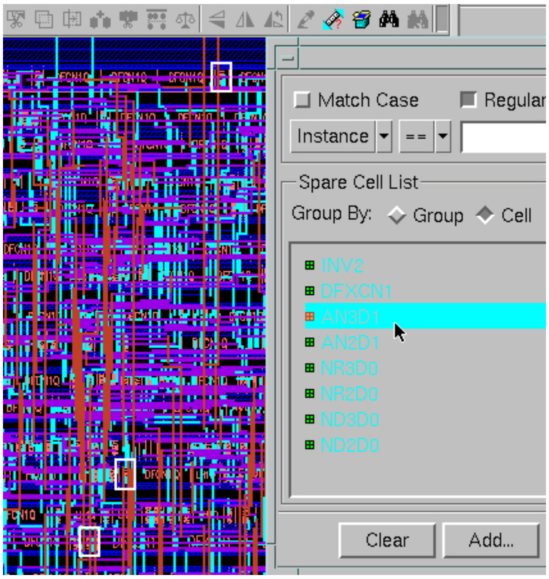


Figure: Highlight Spare Cells in Layout Pane

To expand the spare cell group, simply click the plus icon.

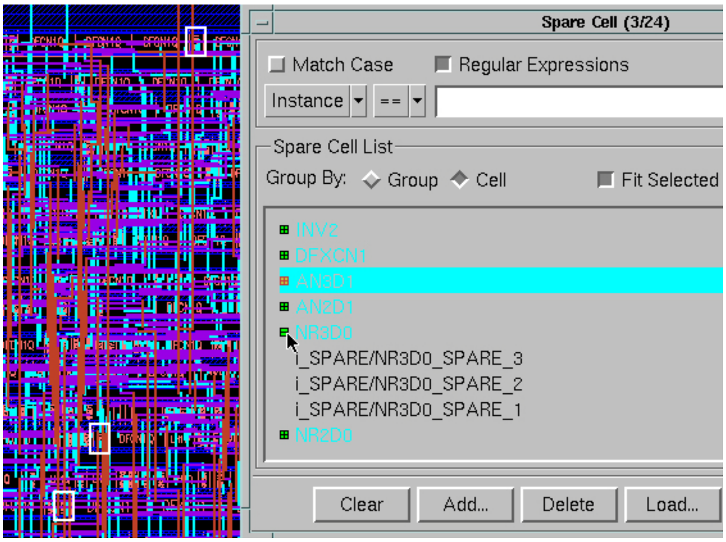


Figure: Expand Spare Cell Group

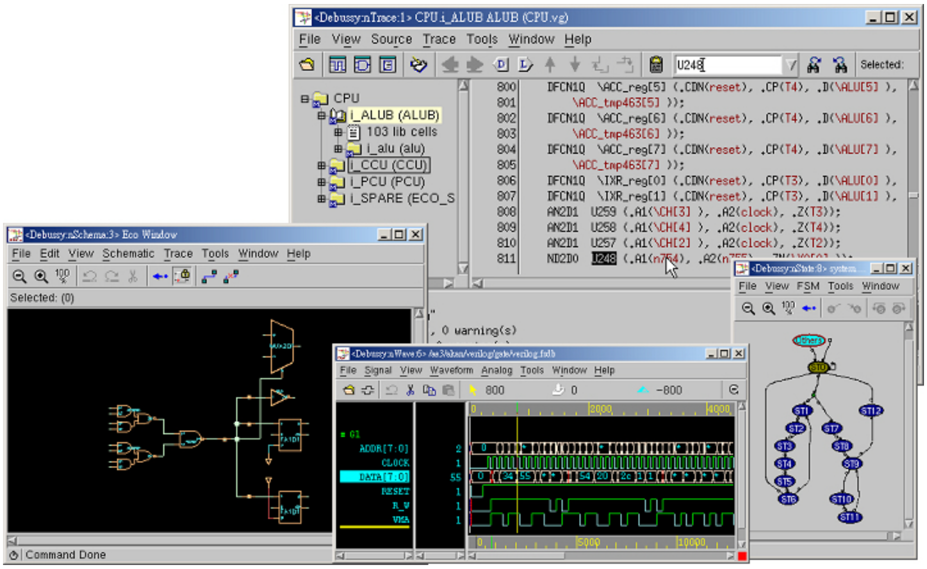
To store the above spare cell information, click the **Save** button, give a desired cell name and click the **OK** button to create a file with the spare cell list information.

Novas nECO Integration

Overview

This chapter describes the details on how Laker works together with the Verdi™ Automated Debug System and related nECO™ Automated Netlist Modification module and performs the ECO function.

Refer to the *Verdi Command Reference Manual* or *nECO User's Guide and Tutorial* and other related documentation for further Verdi and nECO operations.


<Debussy:nTrace:1> CPU.i_ALUB ALUB (CPU.vg)
File View Source Trace Tools Window Help
U248
Selected:
CPU
i_ALUB (ALUB)
103 lib cells
i_alu (alu)
i_CCU (CCU)
i_PCU (PCU)
i_SPARE (ECO_S
<Debussy:nSchema:3> Eco Window
File Edit View Schematic Trace Tools Window Help
Selected: (0)
Command Done
<Debussy:nWave:6>
File Signal View Waveform Analog Tools Window Help
800
0
-800
G1
ADDR[7:0]
CLOCK
DATA[7:0]
RESET
R_W
VMA
<Debussy:nState:8> system
File View FSM Tools Window
Others
ST0
ST1
ST2
ST3
ST4
ST5
ST6
ST7
ST8
ST9
ST10
ST11
ST12

```
DFCN1Q  \ACC_reg[5] (.CDN(reset), .CP(T4), .D(\ALU[5] ),
    \ACC_tmp463[5] ));
DFCN1Q  \ACC_reg[6] (.CDN(reset), .CP(T4), .D(\ALU[6] ),
    \ACC_tmp463[6] ));
DFCN1Q  \ACC_reg[7] (.CDN(reset), .CP(T4), .D(\ALU[7] ),
    \ACC_tmp463[7] ));
DFCN1Q  \IXR_reg[0] (.CDN(reset), .CP(T3), .D(\ALU[0] ),
DFCN1Q  \IXR_reg[1] (.CDN(reset), .CP(T3), .D(\ALU[1] ),
AN2D1  U259 (.A1(\CH[3] ), .A2(clock), .Z(T3));
AN2D1  U258 (.A1(\CH[4] ), .A2(clock), .Z(T4));
AN2D1  U257 (.A1(\CH[2] ), .A2(clock), .Z(T2));
ND2D0  U248 (.A1(n754), .A2(n
```

```
, 0 warning(s)
```

Figure: Novas/Debussy Product

Drag-and-Drop Operation

The drag-and-drop operation is the basic X window mouse operation. The Verdi system uses this mouse operation to integrate its components (nWave, nTrace, nState, nSchema and nECO) by single mouse click and to share selected data.

Laker uses the same mouse operation and works together with the Verdi system.

This integration will focus on the gate-level HDL design with post P&R physical layout database from the LEF/DEF or the stream GDSII data. The RTL design with physical layout database is not included in this integration flow.

The basic flow for the drag-and-drop operation is as follows:

1. Select the desired objects on the active window, which can be the *Eco* window or the Laker *Layout* window.
2. Drag the objects from the active window.
3. Drop the objects to the target window.
4. Change the zoom ratio and highlight the correspondent objects on the target window.

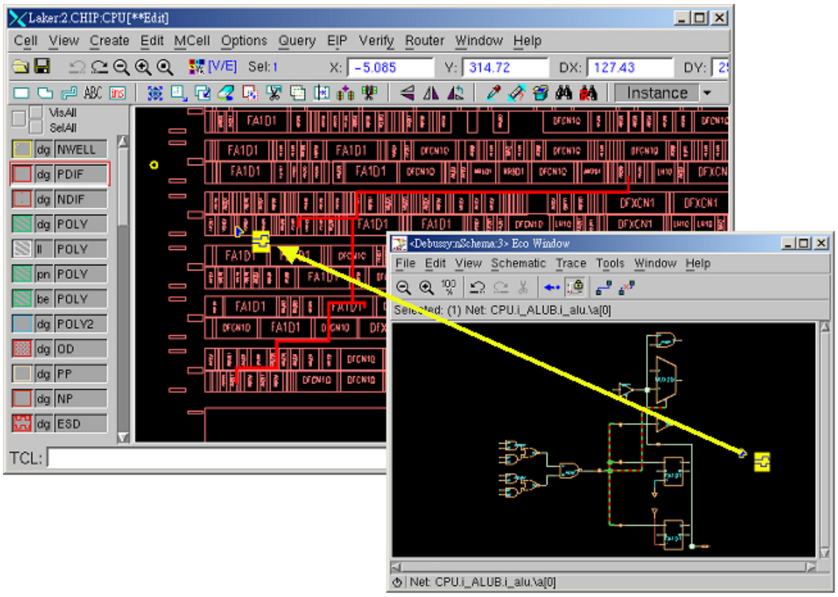
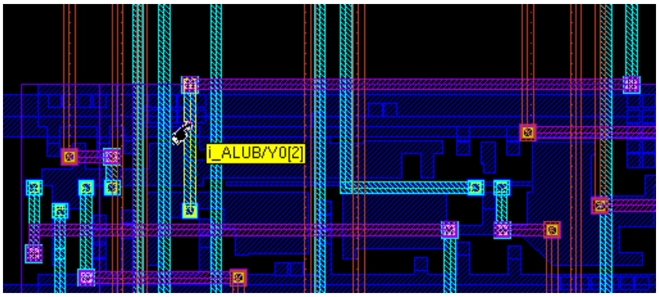

Laker:2 CHIP:CPU[**Edit]
Cell View Create Edit MCell Options Query EIP Verify Router Window Help
[V/E] Sel: 1
X: -5.085
Y: 314.72
DX: 127.43
DY:
Instance
VisAll
SelAll
dg NWELL
dg PDIF
dg NDIF
dg POLY
ll POLY
pn POLY
be POLY
dg POLY2
dg OD
dg PP
dg NP
dg ESD
TCL:
<Debussy:nSchema:3> Eco Window
File Edit View Schematic Trace Tools Window Help
Selected: (1) Net: CPU.i_ALUB.i_alu.\a[0]
Net: CPU.i_ALUB.i_alu.\a[0]

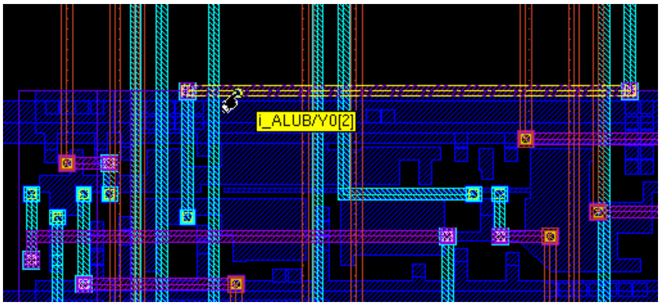
Figure: Drag-and-Drop Operation

The drag-and-drop operation is designed for sharing knowledge between selected objects. In this integration, the complete design hierarchy of the selected objects are packed and passed from active window to target window. This means the drag-and-drop action does not work well if the design hierarchy between active and target windows is different.

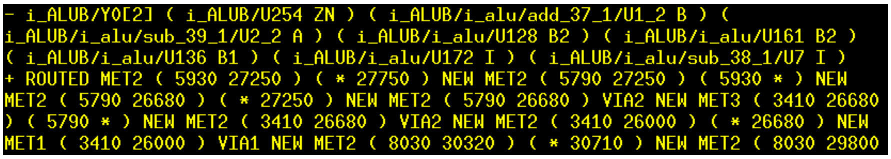
Let's see the difference of design hierarchies in the physical layout.

Through the **Highlight Net** function, the logical net name and instance name will be shown on the screen in a yellow tip window. The following two figures show the same net name on the net in different segments (in different layers). The hierarchical net name is: `I_ALUB/Y0[2]`.





This net is driven by the `ZN` port of `I_ALUB/U254` instance and it drives 7 instances. The partial DEF information is shown as follows:

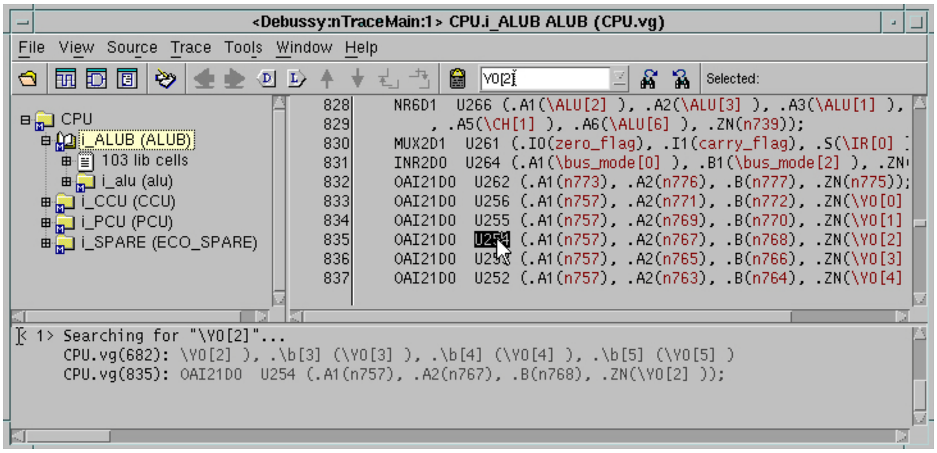


```
- i_ALUB/Y0[2] ( i_ALUB/U254 ZN ) ( i_ALUB/i_alu/add_37_1/U1_2 B ) (
i_ALUB/i_alu/sub_39_1/U2_2 A ) ( i_ALUB/i_alu/U128 B2 ) ( i_ALUB/i_alu/U161 B2 )
( i_ALUB/i_alu/U136 B1 ) ( i_ALUB/i_alu/U172 I ) ( i_ALUB/i_alu/sub_38_1/U7 I )
+ ROUTED MET2 ( 5930 27250 ) ( * 27750 ) NEW MET2 ( 5790 27250 ) ( 5930 * ) NEW
MET2 ( 5790 26680 ) ( * 27250 ) NEW MET2 ( 5790 26680 ) VIA2 NEW MET3 ( 3410 26680
) ( 5790 * ) NEW MET2 ( 3410 26680 ) VIA2 NEW MET2 ( 3410 26000 ) ( * 26680 ) NEW
MET1 ( 3410 26000 ) VIA1 NEW MET2 ( 8030 30320 ) ( * 30710 ) NEW MET2 ( 8030 29800
```

Figure: Partial Net Section of I_ALUB/Y0[2] Net

Let's see how the Verdi system keeps the logical hierarchical path of the same net.

Click the `i_ALUB` instance in the design tree window and find the net name, `Y0[2]`, in the source code window. Find the U254 instance and make sure that the `ZN` port drives the `Y0[2]` net. It is an `OAI21D0` cell.

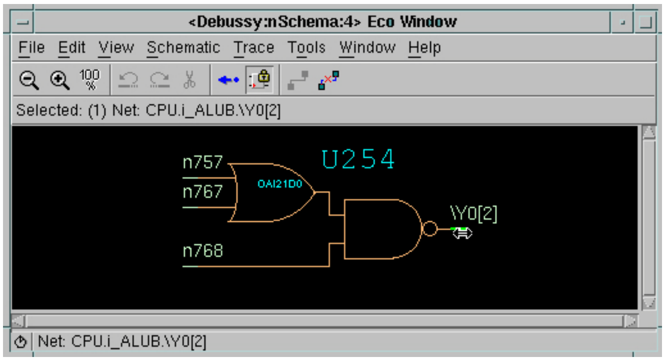


```
828    NR6D1   U266 (.A1(\ALU[2] ), .A2(\ALU[3] ), .A3(\ALU[1] ),
829          , .A5(\CH[1] ), .A6(\ALU[6] ), .ZN(n739));
830    MUX2D1   U261 (.I0(zero_flag), .I1(carry_flag), .S(\IR[0] ]
831    INR2D0   U264 (.A1(\bus_mode[0] ), .B1(\bus_mode[2] ), .ZN
832    OAI21D0   U262 (.A1(n773), .A2(n776), .B(n777), .ZN(n775));
833    OAI21D0   U256 (.A1(n757), .A2(n771), .B(n772), .ZN(\Y0[0]
834    OAI21D0   U255 (.A1(n757), .A2(n769), .B(n770), .ZN(\Y0[1]
835    OAI21D0   U254 (.A1(n757), .A2(n767), .B(n768), .ZN(\Y0[2]
836    OAI21D0   U253 (.A1(n757), .A2(n765), .B(n766), .ZN(\Y0[3]
837    OAI21D0   U252 (.A1(n757), .A2(n763), .B(n764), .ZN(\Y0[4]
```

```
< 1> Searching for "\Y0[2]"...
    CPU.vg(682): \Y0[2] ), .\b[3] (\Y0[3] ), .\b[4] (\Y0[4] ), .\b[5] (\Y0[5] )
    CPU.vg(835): OAI21D0  U254 (.A1(n757), .A2(n767), .B(n768), .ZN(\Y0[2] ));
```

Figure: nTrace Window with Search Result for Y0[2] Net

Select the `U254` instance and invoke the *nECO* window to show this instance as follows:


<Debussy:nSchema:4> Eco Window
File Edit View Schematic Trace Tools Window Help
100 %
n757
n767
n768
U254
OAI21D0
\Y0[2]

Selected: (1) Net: CPU.i_ALUB.\Y0[2]

Net: CPU.i_ALUB.\Y0[2]

Figure: nECO Window with Selected Net, \Y0[2]

The full design hierarchical net name for the net `\Y0[2]` is `CPU.i_ALUB.\Y0[2]`.

To integrate the verification and layout tools, the following problems need to be addressed:

- Top cell name
- Special characters in the Verdi side and the Laker side
- Hierarchical delimiter character
- Performance issue for the database traversing and highlighting

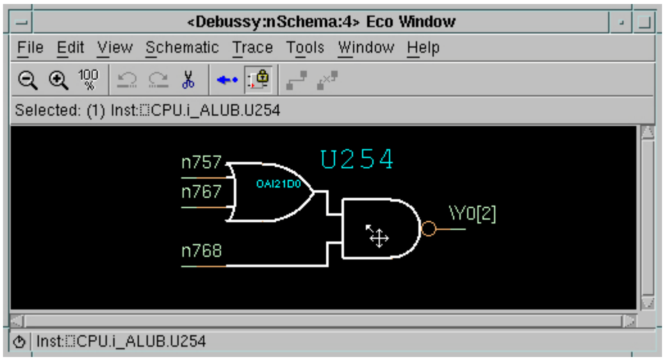

<Debussy:nSchema:4> Eco Window
File Edit View Schematic Trace Tools Window Help
Selected: (1) Inst:CPU.i_ALUB.U254
n757
n767
n768
U254
OAI21D0
\Y0[2]
Inst:CPU.i_ALUB.U254

Figure: nECO Window with Selected Instance, U254

Synopsys makes the following information consistent in both the Laker and Verdi/nECO sides:

- In Laker, the cell name of the *Layout* window is also a part of the design hierarchy.
- In this case, the `i_ALUB/Y0[2]` net under the `CPU` cell is packed as a complete design hierarchy, `CPU/I_ALUB/Y0[2]`, in Laker.
- Drag this net from Laker and drop to Verdi/nECO.
- The Verdi system parses the hierarchy tree and gets the tokens as design hierarchy `CPU.i_ALUB`, and the net name `\Y0[2]`.
- The complete design hierarchy of the same net in Verdi/nECO is `CPU.i_ALUB.\Y0[2]`.
- The search engine in the Verdi system tries to find this net and shows it on the *nECO* window if the net exists.

The complete integration flow is illustrated as follows.

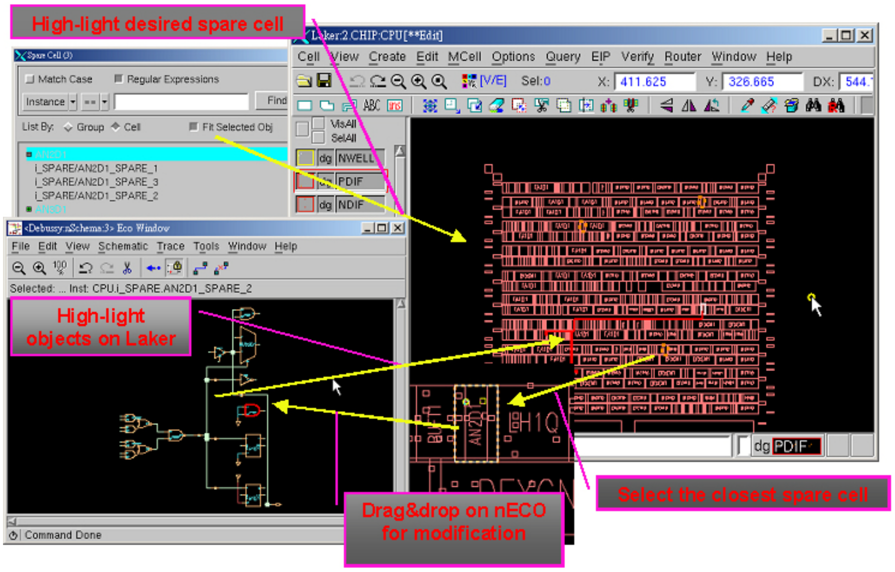

High-light desired spare cell
High-light objects on Laker
Drag&drop on nECO for modification
Select the closest spare cell

Figure: Laker/nECO Co-work Flow

To do the ECO job on front-end gate-level netlist, the IC designer can easily find the right spare cells to meet the timing constraints with less effort.

A README file is provided to guide you through the flow to build the environment for Verdi/nECO and the procedures to go through the Laker/nECO integration. You may get the file (*nECO_Laker.txt*) under your working directory.

NOTE: The symbol library used in this demo case is built with Verdi 5.2. Please change the symbol library to meet your current Verdi version. If you have any symbol library related problem, please contact your local support team for assistance.

Appendix A: CDL Syntax

Overview

This appendix describes the format of CDL syntax supported in Laker with the following sections:

- Commands
- Circuit Elements
- Others

Commands

The control commands include the following:

- .END
- .ENDS
- .GLOBAL
- .INCLUDE
- .MODEL
- .OPTIONS
- .PARAM
- .SUBCKT
- *.FLAT
- *.PININFO
- *.REVERSE

.END

The `.END` line specifies the last statement in the netlist file. The period (.) is an integral part of the name.

Syntax

```
.END
```

.ENDS

The `.ENDS` line specifies the last statement for any subcircuit definition. `<subname>` is the subcircuit name that indicates the subcircuit definition to be terminated.

Syntax

```
.ENDS <subname>
```

or

```
.EOM <subname>
```

where `.EOM` is equivalent to `.ENDS`.

.GLOBAL

The `.GLOBAL` line assigns a global signal.

Syntax

```
.GLOBAL node<:nodetype> ....
```

or

```
*.GLOBAL node<:nodetype> ....
```

where `nodetype` can be `P` or `G` (`P`:Power, `G`:ground). The default power signal is `VDD` and `VCC`; and the default ground signal is `VSS`, `GND` and `0`.

`*.GLOBAL` is equivalent to `.GLOBAL`.

.INCLUDE

Generally, portions of the circuit descriptions will be reused in several files, particularly with common models and subcircuits. In any netlist file, the `.INCLUDE` line is used to include another netlist file.

`.INC` is equivalent to `.INCLUDE`.

Syntax

```
.INCLUDE filename
```

or

```
.INCLUDE '<filepath>/filename'
```

or

```
.INCLUDE "<filepath>/filename"
```

or

```
.INC filename
```

or

```
.INC '<filepath>/filename'
```

or

```
.INC "<filepath>/filename"
```

Examples

```
.INCLUDE /user/test.sp
.INCLUDE '/user/test.sp'
.INCLUDE "/user test.sp"
```

.MODEL

The `.MODEL` line defines a set of device model parameters on a seperate line and assign it with a unique model name. The device element lines refer to the model name. Each device element line contains the device name, the nodes to which the device is connected, and the device model name.

Syntax

```
.MODEL mname type <VERSION=VERSION_NUM> <parname1=val1 parname2=val2 ...>
```

or

```
.MODEL mname type ( <VERSION=VERSION_NUM> <parname1=val1 parname2=val2
...> )
```

where `mname` is the model name, and the `type` is one of the following:

- C : Capacitor model
- D : Diode model
- L : Inductor model
- NMOS : N-channel MOSFET model
- NPN : NPN BJT modle
- PMOS : P-channel MOSFET model
- PNP : PNP BJT model
- R : Resistor model

Parameter values are defined by appending the parameter name followed by an equal sign (=) and the parameter value.

.OPTIONS

The `.OPTIONS` line allows users to reset program control and user options for specific simulation purposes.

Syntax

```
.OPTIONS SCALE=scale DEFW=defw DEFL=defl ....
```

or

```
.OPTION SCALE=scale DEFW=defw DEFL=defl ....
```

where `DEFW` is the default value of MOS channel width. If `DEFW` is not assigned, the `DEFW` is 100u. `DEFL` is the default value of MOS channel length. If `DEFL` is not assigned, the `DEFL` is 100u. `scale` sets the multiplier value for the width and length of elements M, Q, R, C and D.

.PARAM

The `.PARAM` line specifies a parameter. If you don't use an apostrophe (`'`) for expression, the blank is not allowed in the expression.

Syntax I

```
.PARAM parname='value|expression'
```

or

```
.PARAM parname=value|expression
```

Example I

```
.PARAM A=100 B='A+2*3'
```

Syntax II

```
.PARAM funcname(argv1,...)='expression'
```

or

```
.PARAM funcname(argv1,...)=expression
```

Example II

```
.PARAM f(x,y)='x*y+2*x+3*y' g(x,y)=x*y+2
```

.SUBCKT

The `.SUBCKT` line specifies the first statement in the netlist file. The period (.) is an integral part of the name.

Syntax

```
.SUBCKT outputNode .... </> inputNode .... <parname=value> ....
```

or

```
.MACRO outputNode .... </> inputNode .... <parname=value> ....
```

where `.SUBCKT` and `.MACRO` are equivalent.

*.FLAT

Syntax

```
*.FLAT
```

or

```
*.FLAT STOP=subname1 subname2 subname3
```

or

```
*.FLAT LEVEL=level
```

where

`*.FLAT` will flatten all hierarchies from the top subckt;

`*.FLAT STOP` will flatten all hierarchies from the subckt exclusive;

`*.FLAT LEVEL` will flatten to the assigned layer from the subckt;

`*.FLAT STOP=NAND NOR INV` will flatten all hierarchies but stop in the subckt `NAND`, `NOR`, and `INV`;

`*.FLAT LEVEL=3` will flatten three layers from the top.

*.PININFO

The `*.PININFO` line assigns the pin direction.

Syntax

```
*.PININFO node1:pinType node2:pinType .....
```

where `pinType` can be the one of the folowing:

- B: Input/Output
- I: Input
- O: Output

During CDLIn conversion, `*.pininfo xxx:p` can be recognized as `.global xxx:p` while `*.pininfo xxx:g` can be recognized as `.global xxx:g`.

During CDLOut conversion, the port direction (input, output, inout) in `*.pininfo` can be exported. For example:

```
*.pininfo X:I Y:O Z:B         (I:Input  O:Output  B:InOut)
```

Example

```
.SUBCKT INV Z A
*.PININFO Z:O A:I
$subcircuit  contents
.ENDS
```

*.REVERSE

The `*.REVERSE` line swaps the width and length of MOS channel.

If you use `*.REVERSE`, the syntax is as follows.

```
Mxxx md ng ns <nb> mname <l <w>> <other options>
```

`*.REVERSE` has no effect if you specify `W=` and `L=` in MOSFET statement.

Without `*.REVERSE`

```
M1 1 2 3 VDD PMOS 2u 4u , where width=2u length=4u
```

With `*.REVERSE`

```
M1 1 2 3 VDD PMOS 2u 4u , where width=4u length=2u
```

Example

```
*.REVERSE
.SUBCKT INV OP A
M0 OP A VSS VSS nmos 0.5u 2u
M1 OP A VDD VDD pmos 0.5u 2u
.ENDS
```

Circuit Elements

The circuit elements include the following:

- MOSFET
- BJT
- Resistor
- Capacitor
- Inductor
- Diode
- Subcircuit Call

MOSFET

Syntax

```
Mxxx md ng ns <nb> mname <W=w> <L=l> <M=multiplier> .... <$LDD[type]>
<$SKIPPREFIX> <$GROUP=name:<purpose>:<pattern>>
```

or

```
Mxxx md ng ns <nb> mname <w <l>> <other options>
```

where `md`, `ng`, and `ns` are the drain, gate, and source nodes, respectively. `<nb>` is the optional bulk substrate nodes. `mname` is the model name. `L` and `W` are the channel length and width. `M` is the multiply factor.

`$SKIPPREFIX` ignores the first character of the element name.

`$GROUP` defines the group name, group purpose and pattern name.

In addition, the model name will be removed if `$LDD` exists. The rule is follows:

With `$LDD`:

If `mname` (or `type`) starts with `N` or `n`: `LDDN`

If `mname` (or `type`) starts with `P` or `p`: `LDDP`

If `mname` (or `type`) starts with `E` or `e`: `LDDE`

If `mname` (or `type`) starts with `D` or `d`: `LDDD`

Otherwise: `LDD`

Example

```
$ create element name ABC for MABC.
MABC VDD A O VDD PMOS W=2u L=2u $SKIPPREFIX
```

```
$ show group name (Mch3) in the design hierarchy browser
mm14 n1n77 n1n77 vdd vdd p l=4u w=12u m=4 $group=Mch3:match

$ A mosfet M1 with drain, gate, source and bulk connected to
$ node A, B, C and D; and the width and length of the gate are
$ given as 1 and 2 microns.
M1 A B C D NMOS W=1u L=2u

$ The model name should be LDDN
MI1 n1 n2 n3 n4 N L=4U W=10U AS=14P AD=115.2P PS=22.8U PD=46.4U M=1 $LDD

$ The modle name should be LDD[NKA]
MI1 n1 n2 n3 n4 N L=4U W=10U AS=14P AD=115.2P PS=22.8U PD=46.4U $LDD[NKA]
```

BJT

Syntax

`Qxxx nc nb ne <ns> mname <AREA=area> <M=multiplier> .... <$EA=value> <$W=w> <$L=l> <$SKIPPREFIX> <$GROUP=name:<purpose>:<pattern>>`

or

`Qxxx nc nb ne <ns> mname <area> <other options>`

where `nc`, `nb`, and `ne` are the collector, base, and emitter nodes, respectively. `<ns>` is the optional substrate node. `mname` is the model name. `AREA` is the area factor, which determines the number of equivalent parallel devices of a specified model. `$L` and $`W` are the channel length and width. `M` is the multiply factor.

`$SKIPPREFIX` ignores the first character of the element name.

`$GROUP` defines the group name, group purpose and pattern name.

Example

```
$ A bipolar Q1 with collector, base, emitter and substrate
$ connected to node A, B, C and D; and emitter area is 100.
Q1 A B C D PNP area=100
```

Resistor

Syntax

`Rxxx n1 n2 <mname> <R=r> <TC1=tc1> <TC2=tc2> <SCALE=scale> <M=multiplier> .... <$SUB=substrate> <$[mname]|$.MODEL=mname> <$W=w> <$L=l> <$NS=ns> <$SEGMENT=ns> <$SKIPPREFIX> <$GROUP=name:<purpose>:<pattern>>`

or

`Rxxx n1 n2 <mname> <R=r> <TC1=tc1> <TC2=tc2> <W=w> <L=l> <SCALE=scale> <M=multiplier> .... <$SUB=substrate> <$[mname]|$.MODEL=mname> <$NS=ns> <$SEGMENT=ns> <$SKIPPREFIX> <$GROUP=name:<purpose>:<pattern>>`

or

```
Rxxx n1 n2 <mname> <r <tc1 <tc2>>> <other options>
```

where `n1` and `n2` are the two element nodes. `mname` is the model name. `R` is the resistance. `TC1` and `TC2` are the first and second order temperature coeffs, respectively. `L/$L` and `W/$W` are the channel length and width. `SCALE` is the scale factor. `M` is the multiply factor.

`$W` is the Wiegth of Resistor.

`$L` is the length of Resistor.

`$SUB` is the substrate of Resistor.

`$NS` or `$SEGMENT` is the segment number of Resistor.

`$SKIPPREFIX` ignores the first character of the element name.

`$GROUP` defines the group name, group purpose and pattern name.

Example

```
$ A resistor R1 is connected to node A and node B with
$ a resistance of 100 ohms.
R1 A B RES R=100

R2 C D RES $W=1u $L=50u $NS=5 $sub=vdd $group=mch_R:match:ABABABABAB
R3 E F RES $W=1u $L=50u $NS=5 $sub=vdd $group=mch_R:match:ABABABABAB
```

Capacitor

Syntax

```
Cxxx n1 n2 <mname> <C=c> <TC1=tc1> <TC2=tc2> <W=w> <L=l> <SCALE=scale>
<M=multiplier> .... <$SUB=substrate> <$[mname]|$.MODEL=mname> <$W=w>
<$L=l> <$SKIPPREFIX> <$GROUP=name:<purpose>:<pattern>>
```

or

```
Cxxx n1 n2 <mname> <c <tc1 <tc2>>> <other options>
```

where `n1` and `n2` are the two element nodes. `mname` is the model name. `C` is the capacitance. `TC1` and `TC2` are the first and second order temperature coeffs, respectively. `L` and `W` are the channel length and width. `SCALE` is the scale factor. `M` is the multiply factor.

`$W` is the Wiegth of Capacitor.

`$L` is the length of Capacitor.

`$SUB` is the substrate of Resistor.

`$SKIPPREFIX` ignores the first character of the element name.

`$GROUP` defines the group name, group purpose and pattern name.

Example

```
$ A capacitor C1 is connected to node A and B with
$ a capacitance of 100 Pico farads.
C1 A B CAP C=100p

C2 C D CAP C=100p $W=20u $L=20u $sub=vdd $group=mch_cap:match:ABCCBA
C3 C D CAP C=100p $W=20u $L=20u $sub=vdd $group=mch_cap:match:ABCCBA
```

Inductor

Syntax

```
Lxxx n1 n2 <mname> <L=inductance> <TC1=tc1> <TC2=tc2> <SCALE=scale>
<M=multiplier> .... <$SUB=substrate> <$SKIPPREFIX>
<$GROUP=name:<purpose>:<pattern>>
```

or

```
Lxxx n1 n2 <mname> <inductance <tc1 <tc2>>> <other options>
```

where `n1` and `n2` are the two element nodes. `mname` is the model name. `L` is the inductance. `TC1` and `TC2` are the first and second order temperature coeffs, respectively. `SCALE` is the scale factor. `M` is the multiply factor.

`$SKIPPREFIX` ignores the first character of the element name.

`$GROUP` defines the group name, group purpose and pattern name.

Example

```
$ An inductor L1 is connected to node A and node B with
$ an inductance of 1 micro-henries.
L1 A B L=1u
```

Diode

Syntax

```
Dxxx positive minus mname <AREA=area> <PJ=pj> <W=w> <L=l> <M=multiplier>
.... <$SUB=substrate> <$SKIPPREFIX> <$GROUP=name:<purpose>:<pattern>>
```

or

```
Dxxx positive minus mname <area <pj>> <other options>
```

where `positive` and `minus` are the positive and negative nodes, respectively. `mname` is the model name. `L` is the inductance. `AREA` is the area factor. `L` and `W` are the channel length and width. `M` is the multiply factor.

`$SKIPPREFIX` ignores the first character of the element name.

`$GROUP` defines the group name, group purpose and pattern name.

Example

```
$ An inductor D1 is connected to node A and node B;
$ and the area is 100.
D1 A B DIODE AREA=100
```

Subcircuit Call

Subcircuit is used by specifying pseudo-element beginning with the letter X, followed by the circuit nodes to be used in expanding the subcircuit.

Syntax

```
Xxxx name.... </> subname <M=multiplier> <parname=value> ... <$SKIPPREFIX>
<$GROUP=name:<purpose>:<pattern>>
```

or

```
Xxxx subname <M=multiplier> $PINS pin=node .... <$SKIPPREFIX>
<$GROUP=name:<purpose>:<pattern>>
```

where `subname` is the subcircuit name, and `$PINS` is used to make the subcircuit call to map node by name (by default, it is mapped by position).

`$SKIPPREFIX` ignores the first character of the element name.

`$GROUP` defines the group name, group purpose and pattern name.

Examples

```
.SUBCKT SSS A B C
$subcircuit  contents
.ENDS
```

The following statements are equivalent.

```
X1 1 2 3 SSS
X1 SSS $PINS A=1 B=2 C=3
```

M Factor Expansion

If X-Type element with multiplier, the element is expanded according to the multiplier value.

Examples

```
X1 A B C SSS M=4
$ We will expand X1 to X1::0, X1::1, X1::2 and X1::3.
X1::0 A B C SSS
X1::1 A B C SSS
X1::2 A B C SSS
X1::3 A B C SSS
```

Subckt Expansion by Parameter Value

Expand the element according to the parameter value.

Examples

```
.SUBCKT NAND2 Z A B WN=1u
$subcircuit  contents
.ENDS
X1 O I1 I2 NAND2 WN=2u
X2 O I1 I2 NAND2 WN=3u
X3 O I1 I2 NAND2 WN=4u
X4 O I1 I2 NAND2 WN=2u
```

Since `X1`, `X2` and `X3` pass different parameter value of `WN` to subkct `NAND2`, the subckt `NAND2` is expanded to `NAND2_0` (for `X1`) , `NAND2_1` (for `X2`) and `NAND2_2` (for `X3`).

`X1` and `X4` have the same parameter value, `X4` will use `NAND2_0` as its master.

Others

Others supported syntax format:

- Built-in Functions
- Scale Factor
- Comment

Built-In Functions

sin(x): sine

cos(x): cosine

tan(x): tangent

asin(x): arc sine

acos(x): arc cosine

atan(x): arc tangent

sinh(x): hyperbolic sine

cosh(x): hyperbolic cosine

tanh(x): hyperbolic tangent

abs(x): absolution value. return |x|.

sqrt(x): square root. return (sign of x)sqrt(|x|).

pow(x,y): absolute power. return x^int(y).

pwr(x,y): signed power. return (sing of x)|x|^y.

log(x): natural logarithm. return (sign of x)log(|x|).

log10(x): base 10 logarithm. return (sign of x)log10(|x|).

exp(x): exponential.

db(x): decibels. return (sign of x)20log10(|x|).

int(x): integer. return the integer portion of x.

sgn(x): return sign. return -1 if x<0. return 0 if x=0. return 1 if x>0.

sign(x,y): transfer sign. return (sign of y)|x|.

min(x,y): smaller of two args.

max(x,y): larger of two args.

Scale Factor

T	1E12
G	1E9
MEG	1E6
X	1E6
K	1E3
%	1E-2
M	1E-3
MIL	25.4E-6
U	1E-6
N	1E-9
P	1E-12
F	1E-15

Comment

Comment lines may be placed anywhere in the circuit description. The asterisk (`*`) is used as the first non-blank character. The inline dollar sign (`$`) indicates that this line is a comment statement.

Syntax

```
* <any comment>
```

or

```
$ <any comment>
```

Example

```
* This is a comment line
$ This is a comment line too.
R1 1 2 R=2   $ This is a resistor
```

In the comment line, you may also define a special parameter `$segment` for UDD device. The parameter is saved into a parameter list with name `$segment`, which is case-insensitive in the SPICE file. For example,

```
r1 A B 50 $segment=5
```

The character `$` is preserved for output the original spice format; and thus the parameter name must be defined with comment character `$`. This makes it different from the parameter `segment`.

If you want to map the parameter `$segment` to the parameter of parameterized device, you must change its name as `__segment`. For example,

```
M pmos  Lib1 Pdevice  {} {set A [expr $L*1e+6]; set seg [expr $__segment*2]}
```

Appendix B: LEF Syntax

Overview

This appendix documents the supported LEF v5.5 Syntax.

Supported LEF Syntax

```
VERSION number ;
NAMESCASESENSITIVE {ON | OFF} ;
 [ UNITS
  [DATABASE MICRONS LEFconvertFactor ;]
END UNITS]


{ LAYER (Nonrouting) statement | LAYER (Routing) statement } ...

# Cut Layer
LAYER layerName
  TYPE CUT ;
END layerName

# Implant Layer
LAYER layerName
  TYPE IMPLANT ;
  [WIDTH minWidth ;]
END layerName

# Masterslice or Overlap Layer
LAYER layerName
  TYPE {MASTERSLICE | OVERLAP} ;
END layerName

# Routing Layer
LAYER layerName
  TYPE ROUTING ;
  DIRECTION {HORIZONTAL | VERTICAL} ;
  PITCH distance ;
  WIDTH defWidth ;
  [OFFSET distance ;]
END layerName

{ VIA viaName
    [FOREIGN foreignCellName [pt [orient]];]
    {LAYER layerName ;
       {RECT pt pt ;} ...} ...
END viaName } ...

[ NONDEFAULTRULE rulename1
```

```
    {LAYER layerName
       WIDTH width ;
       SPACING minSpacing ;
       [WIREEXTENSION value ;]
       [RESISTANCE RPERSQ value ;]
       [CAPACITANCE CPERSQDIST value ;]
       [EDGECAPACITANCE value ;]
    END layerName} ...
    {VIA viaStatement} ...
    [SPACING
       [SAMENET layerName layerName minSpace [STACK] ;] ...
    END SPACING]
END] ...

{ MACRO macroName
    [CLASS
      { COVER [BUMP]
      | RING
      | BLOCK [BLACKBOX]
      | PAD [INPUT | OUTPUT |INOUT | POWER | SPACER | AREAIO]
      | CORE [FEEDTHRU | TIEHIGH | TIELOW | SPACER | ANTENNACELL]
      | ENDCAP {PRE | POST | TOPLEFT | TOPRIGHT | BOTTOMLEFT
               | BOTTOMRIGHT}
      }
    ;]
    [FOREIGN foreignCellName [pt [orient]] ;]
    [ORIGIN pt ;]
    [SIZE width BY height ;]
    [SYMMETRY {X | Y | R90} ... ;]
    [SITE siteName ;]
    [PIN pinName
     [DIRECTION {INPUT | OUTPUT [TRISTATE] | INOUT | FEEDTHRU} ;]
     {PORT
       { LAYER layerName
         [WIDTH width ;]
         { PATH pt ... ;
         | PATH ITERATE pt ... DO numX BY numY STEP spaceX spaceY  ;
         | RECT pt pt ;
         | RECT ITERATE pt pt DO numX BY numY STEP spaceX spaceY  ;
         | POLYGON pt pt pt pt ... ;
        | POLYGON ITERATE pt pt pt pt ... DO numX BY numY STEP spaceX spaceY
;
         } ...
      | VIA pt viaName ;
      | VIA ITERATE pt viaName DO numX BY numY STEP spaceX spaceY  ;
        } ...
       END} ...
    END pinName] ...

    [OBS
      { LAYER layerName [ [WIDTH width ;]
           { PATH pt ... ;
        | PATH ITERATE pt ... DO numX BY numY STEP spaceX spaceY  ;
        | RECT pt pt ;
        | RECT ITERATE pt pt DO numX BY numY STEP spaceX spaceY  ;
        | POLYGON pt pt pt pt ... ;
        | POLYGON ITERATE pt pt pt pt ... DO numX BY numY STEP spaceX spaceY
;
             } ...
      | VIA pt viaName ;
      | VIA ITERATE pt viaName DO numX BY numY STEP spaceX spaceY  ;
        } ...
    END] ...

  END macroName } ...


END LIBRARY
```

Appendix C: DEF Syntax

Overview

This appendix documents the supported DEF v5.5 Syntax.

Supported DEF Syntax

```
VERSION statement
NAMESCASESENSITIVE statement
DIVIDERCHAR statement
BUSBITCHARS statement
DESIGN statement
[ TECHNOLOGY statement ]
[ UNITS statement ]
[ HISTORY statement ]...
[ PROPERTYDEFINITIONS section ]
[ DIEAREA statement ]
[ ROWS statement ] ...
[ TRACKS statement ]...
[ GCELLGRID statement ]...
[ VIAS statement ]
[ REGIONS statement ]
COMPONENTS section
[ PINS section ]
[ PINPROPERTIES section ]
[ BLOCKAGES section ]
[ SLOTS section ]
[ FILLS section ]
[ SPECIALNETS section ]
NETS section
[ SCANCHAINS section ]
[ GROUPS section ]
[ BEGINEXT section ]
END DESIGN statement


================================
VERSION versionNumber ;

NAMESCASESENSITIVE {OFF | ON} ;

DIVIDERCHAR "character" ;

BUSBITCHARS "delimiterPair" ;

DESIGN designName ;

UNITS DISTANCE MICRONS DEFconvertFactor ;
```

```
PROPERTYDEFINITIONS
[objectType propName propType [RANGE min max] [value | stringValue]
;] ...
END PROPERTYDEFINITIONS

objectType =
{DESIGN | COMPONENT | NET | SPECIALNET | GROUP
 | ROW |COMPONENTPIN | REGION}
propType =
{INTEGER | REAL | STRING}

DIEAREA pt pt;

ROW rowName rowType origX origY orient
{ DO numX BY 1 STEP spaceX 0
| DO 1 BY numY STEP 0 spaceY }
[+ PROPERTY {propName propVal}...]...;

TRACKS
{X | Y} start DO numTracks STEP space
LAYER layerName ;

GCELLGRID
{ X start DO numColumns+1 STEP space } ...
{ Y start DO numRows+1 STEP space ; }...


VIAS numVias ;
[- viaName
[+ RECT layerName pt pt]... ;]...
END VIAS


COMPONENTS numComps ;
[- compName modelName
 [+ SOURCE {NETLIST | DIST | USER | TIMING}]
[+ FOREIGN foreignCellName pt orient]
[+ {FIXED pt orient | COVER pt orient | PLACED pt orient
| UNPLACED} ]
 [+ PROPERTY {propName propVal} ...]...
;] ...
END COMPONENTS

PINS numPins ;
[ [- pinName + NET netName]
[+ SPECIAL]
[+ DIRECTION {INPUT | OUTPUT | INOUT | FEEDTHRU}]
[+ USE {SIGNAL | POWER | GROUND | CLOCK | TIEOFF |
ANALOG | SCAN | RESET}]
 [+ LAYER layerName pt pt]
[+ COVER pt orient | FIXED pt orient | PLACED pt orient]
; ] ...
END PINS


PINPROPERTIES num;
[- {compName pinName | PIN pinName}
[+ PROPERTY {propName propVal} ...] ...
;] ...
```

```
END PINPROPERTIES]


BLOCKAGES numBlockages ;
[- LAYER layerName
[+ COMPONENT compName | + SLOTS | + FILLS | + PUSHDOWN]
{RECT pt pt} ...
;] ...
[- PLACEMENT
[+ COMPONENT compName | + PUSHDOWN]
{RECT pt pt} ...
;] ...
END BLOCKAGES

SPECIALNETS numNets ;
[- netName [ ( compNameRegExpr pinName ) ] ...
[+ WIDTH layerName width] ...
 [specialWiring] ...
[+ SOURCE {DIST | NETLIST | TIMING | USER}]
 [+ ORIGINAL netName]
[+ USE {ANALOG | CLOCK | GROUND | POWER | RESET
 | SCAN | SIGNAL | TIEOFF}]
 [+ PROPERTY {propName propVal} ...] ...
;] ...
END SPECIALNETS]


specialWiring=
{+ COVER | + FIXED | + ROUTED | + SHIELD shieldNetName}
layerName width
[+ SHAPE {RING | PADRING | BLOCKRING |
STRIPE | FOLLOWPIN| IOWIRE | COREWIRE
| BLOCKWIRE | BLOCKAGEWIRE | FILLWIRE}]
( x y )
{ ( x * ) | ( * y ) | viaName [DO numX BY numY STEP
stepX stepY]} ...
[NEW layerName width
[+ SHAPE {RING | PADRING | BLOCKRING
| STRIPE | FOLLOWPIN| IOWIRE | COREWIRE
| BLOCKWIRE | BLOCKAGEWIRE | FILLWIRE}]
( x y )
{ ( x * ) | ( * y ) | viaName [DO numX BY numY STEP
stepX stepY]} ...
] ...

NETS numNets ;
[- { netName
[ ( {compName pinName | PIN pinName} }
[+ SHIELDNET shieldNetName ] ...
[+ VPIN vpinName [LAYER layerName] pt pt
[+ NONDEFAULTRULE ruleName]
[regularWiring] ...
[+ SOURCE {DIST | NETLIST | TEST | TIMING | USER}]
 [+ ORIGINAL netName]
[+ USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL
| TIEOFF}]
 [+ PROPERTY {propName propVal} ...] ...
;] ...
END NETS
regularWiring =
{+ COVER | + FIXED | + ROUTED | + NOSHIELD}
layerName [TAPER | TAPERRULE ruleName] ( x y [value] )
{( x * [value] ) | ( * y [value] ) | ( * * [value] ) | viaName} ...
[NEW layerName [TAPER | TAPERRULE ruleName] ( x y [value] )
{( x * [value] ) | ( * y [value] ) | ( * * [value] ) | viaName} ...
] ...


SCANCHAINS numScanChains ;
[- chainName
[+ PARTITION partitionName [MAXBITS maxbits]]
[+ COMMONSCANPINS [ ( IN pin )] [( OUT pin ) ] ]
+ START {fixedInComp | PIN} [outPin]
+ FLOATING
{floatingComp [ ( IN pin ) ] [ ( OUT pin ) ] [ ( BITS numBits ) ]} ...
[+ ORDERED
{fixedComp [ ( IN pin ) ] [ ( OUT pin ) ] [ ( BITS numBits ) ]} ...
] ...
+ STOP {fixedOutComp | PIN} [inPin] ]
;] ...
END SCANCHAINS


END DESIGN designName
```

Index