Automated Custom Physical Design Flow Guide

Product Version 5.0
July 2002

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Contents

1

Introduction to the Automated Custom Physical Design Flow

The automated custom physical design (ACPD) flow is a powerful solution for automating the creation of physical mask layout data using advanced parameterized cells, interactive editing, automated placement, and automated routing, all in a connectivity-based environment. This advanced approach yields the ever-shorter design cycle required for full-custom physical layout in today's high-performance integrated circuits.

The ACPD flow relies upon a methodical approach to maintaining a schematic hierarchy database that coincides with the appropriate physical layout cellview. Many design engineers and layout designers are not accustomed to managing their data in this way.

For methodology information on the ACPD flow, refer to *Appendix A Automated Custom Physical Design: Device Level Methodology Guide*.

ACPD Flow Components

The ACPD flow describes the use of Cadence® software tools to produce integrated circuit physical mask data.

- Cadence Design Framework II

  Design framework II (DFII) provides an integrated design environment for electronic design automation. DFII is the core database for the tools in the ACPD flow.

- Technology file

  The technology file describes the layer attributes, physical layout rules, electrical rules, application-specific rules, and stream translation rules. The technology file also defines technology file devices, such as parameterized cells (pcells), which are used by other applications such as the Virtuoso® XL Layout Editor (Virtuoso XL).

- Reference libraries

  DFII reference libraries contain reusable design components that can be referenced from and shared across project-specific libraries. Reference libraries must be in the current search path defined in the Library Manager. Depending on the design environment, your reference library set could include the Cadence standard libraries (such as `basic` and `analogLib`) as well as site-specific libraries.

  These reference libraries need to contain a complete set of cellviews (the schematic cellview and the corresponding layout and symbol views must be present). If a corresponding layout cellview is not available, Virtuoso XL traverses down the hierarchy until a suitable layout cellview is found.

- SKILL-based parameterized cells using relative object design (ROD) constructs

  Parameterized cells (pcells) are a method for creating reusable physical design components. They are defined using the Cadence SKILL programming language enhanced with ROD SKILL constructs. Pcells can be put in a reference library and shared across multiple projects using the same technology process.

  The pcell parameters can be controlled with Component Description Format (CDF) parameters. CDF parameters describe the attributes of an individual component or libraries of components. CDF parameters provide a single location for storing information, so applications can share and reuse parts of the component description. CDFs can be defined at the library level or at the cell level. When CDFs are defined at the cell level, all views share the same parameter name and value.

  The ROD SKILL constructs allow quick, intuitive, technology-tolerant cell generation. The ROD objects are similar in function to DFII layout shapes. The ROD SKILL functions are similar to DFII SKILL functions. The ROD objects allow many new concepts, including

- System- and user-defined handles

  Handles can be points on any figure's edge, properties on a ROD object (database shape), or x and y locations on any figure's bounding box, as dynamic database objects.

- The naming of shapes

  Makes accessible named objects (and handles) at any level of the hierarchy using a named path.

- The ability to align figures using extracted point handles and technology file–based design rules

- The ability to create a multipart path (MPP)

  A MPP is a composite data object that can be composed of multiple paths and rectangles.

- Easier creation of hierarchical parameterized cells

Some of the techniques used in the ACPD flow require some advanced SKILL-based pcells. Sample SKILL-based pcells are included in the installation of Virtuoso XL.

■ Virtuoso Schematic Composer

The Virtuoso Schematic Composer is a design entry tool that enables logic and circuit design engineers to produce schematics. These schematics supply hierarchical graphical connectivity information that can be transferred into the physical layout design.

■ Virtuoso Layout Editor and Virtuoso XL Layout Editor

The Virtuoso Layout Editor is the editor of the Virtuoso tools. You use the layout editor to prepare custom integrated circuit designs by

- Creating and editing polygons, paths, rectangles, circles, ellipses, donuts, pins, and contacts in layout cellviews

- Placing cells into other cells to create hierarchical designs

- Connecting a pin or group of pins in a net internally or externally

Virtuoso XL extends the Virtuoso layout editor with advanced interactive editing capability and provides a design environment that integrates automated place-and-route functionality. Virtuoso XL combines the schematic connectivity information with the physical layout data to create an automated method for generating physical layout data. Virtuoso XL is compatible with legacy design data. Once pin information has been added to legacy layout design data, Virtuoso XL can use the cells in the connectivity mode.

The connectivity source in the ACPD flow can be a circuit description language (CDL) netlist or a schematic originated in the Virtuoso Schematic Composer. For the purpose of this document, Virtuoso Schematic Composer schematics are the implied connectivity source for the ACPD flow.

Once Virtuoso XL has established the connectivity source for the layout cellview, the Virtuoso tool keeps track of the connectivity on instances and shapes as they are entered in the layout cellview.

The Virtuoso XL layout editor starts with connectivity on pins and nets derived from the connectivity source. As new shapes are connected to the net, the Virtuoso XL extractor propagates the connectivity to the new shape. If a short occurs between two nets in the design, a marker appears on the violation. This guides you in interactive editing to build the design in a correct-by-construction fashion.

- Virtuoso constraint manager

  The constraint manager provides an interface to create and edit physical design constraints used in various components inside the ACPD flow. An inheritance mechanism allows a layout view to automatically inherit all the constraints defined in its connectivity reference. The constraint manager supports net-based, alignment, fixed, distance, grouping, and symmetry constraints.

- Virtuoso custom placer

  The Virtuoso custom placer provides an automated placement solution for cells and devices in custom block building. This placement automation can be mixed-and-matched freely with manual placement performed using interactive editing functionality in Virtuoso XL.

- Virtuoso custom router

  The Virtuoso custom router provides routing solutions for the ACPD flow. The router is a constraint-driven, correct-by-construction shape-based router. The Virtuoso custom router does not have to maintain the minimum width/space-based routing grid, as other routing tools do. The router recognizes many different types of constraints applied to nets, including width, space, length, and electrical constraints.

- Virtuoso Compactor

  The Virtuoso Compactor compacts layout objects as closely as possible to achieve smallest cell size. The design is compacted in such a way that design rules are not violated in the final layout. The compactor supports pcells built with ROD constructs, including multipart paths, as well as advanced interactive editing capabilities such as abutment.

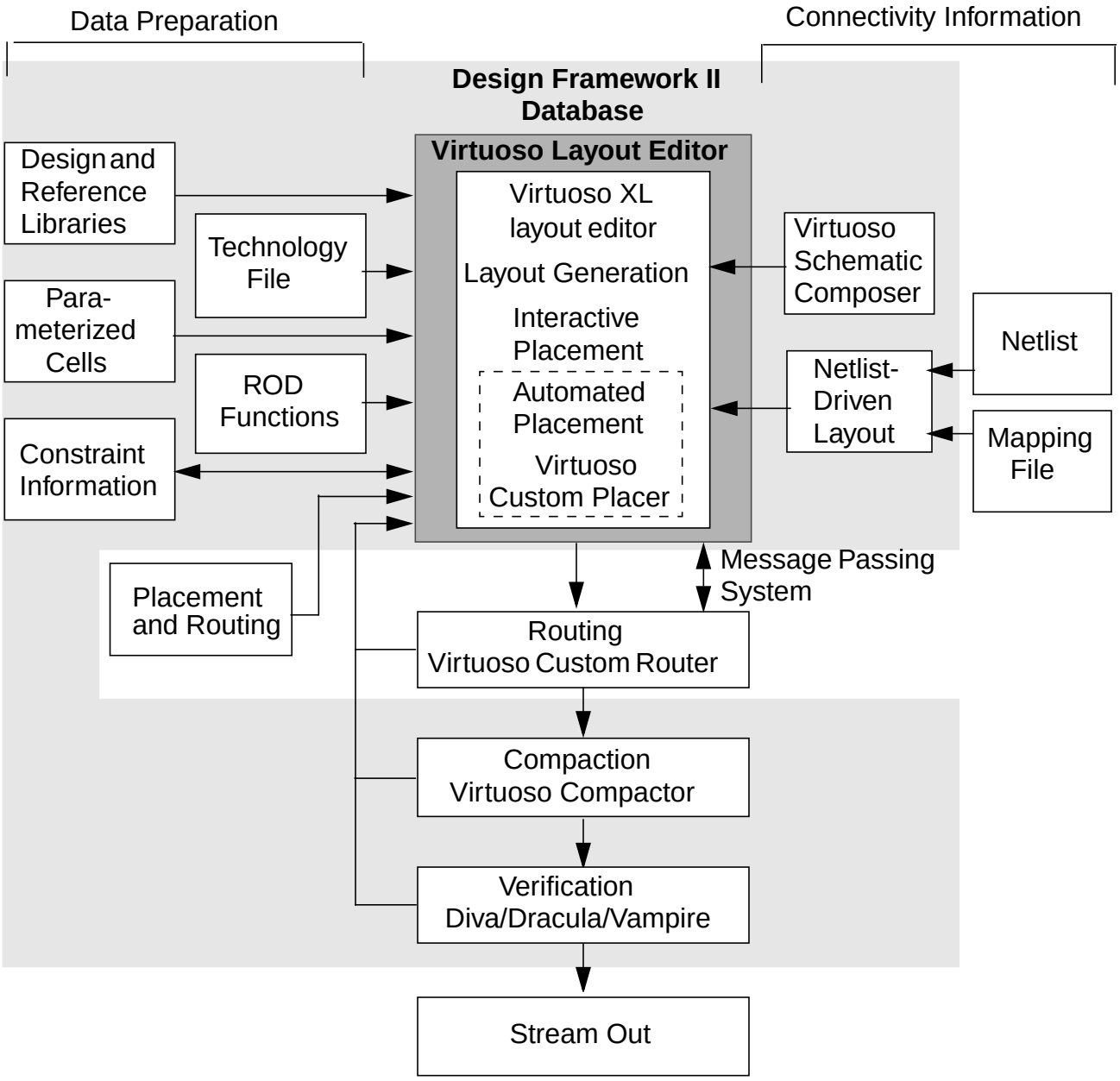
- Assura™ Diva® interactive verification

The Diva interactive verification product lets you find and correct design errors. Using layer processing to prepare data, the Diva verification tool checks physical and electrical design rules. The Diva tool also performs layout- versus-schematic comparisons.

The Diva interactive verification product helps you find errors early in the design process and lets you view them interactively to help speed error diagnosis and correction.

The flow diagram shows how these tools work together.

Automated Custom Physical Design Tool Flow Diagram



Design Cycle for a Device-Level Physical Design

This section describes, in general terms, the full design cycle for a device-level physical design that meets the needs of a custom or semi-custom layout. A subset of the tools in the ACPD flow are used in the design cycle for device-level physical design. Unlike regular digital designs for ASIC chips where synthesis techniques are available, the design and layout of analog/mixed signal blocks and performance-critical digital blocks still require manual, labor-intensive efforts. A well defined methodology is required to ensure a smooth design flow with fewer iterations and a faster turnaround.

Engineering Input

The physical design flow starts with schematics or schematic netlists as the primary input. For this design, the input is from a Virtuoso composer schematic. Topologies for the design at the circuit or device-level are chosen during the circuit design phase. Here, the actual transistor sizes and values for components such as resistors and capacitors are calculated. Throughout high-level modeling, partitioning, and circuit design, functional verification is performed to systematically verify that any decisions being made adhere to the design requirements. This information will be used to estimate the physical size of the design.

Many engineers pay particular attention to the placement of devices on the engineering schematic such that it represents not only the electrical connectivity but communicates their desired physical placement to the layout designer. This duality permits the use of the schematic to help in determining the placement in layout. Top-level constraints, such as area, performance, timing, and reliability requirements, have to be accounted for as well. Physical design should be both connectivity and constraint driven.

It is expected that the engineering drawings will not be writable by the layout designer. In some cases and customer methodologies the layout designer may be permitted to update the schematics. This is a supported, but not required, methodology. To update a schematic the designer will need a composer license.

During the hand-over of the schematics to the layout designer, the final block specifications are agreed upon by both the layout designer and the engineer. To facilitate this process, the process technology information is required. Much of the required technology information is predefined and shipped with the parts library that will be used to implement the design. This information is in the technology file.

Data Preparation

This phase explores various architectures for the block and begins evaluating the top-level specification and process technology–imposed design constraints. Other issues that must be resolved during this phase include

- Process limitations and feasibility of the design's critical blocks for meeting performance specifications
- Device type considerations (MOS transistors, capacitors, resistors, inductors, bipolar transistors, diodes)
- Layout environment expectations. Examples include additional capacitance due to fringe capacitors, general layout parasitics, electrostatic discharge (ESD) considerations, electromigration, hot-carrier effects, short-channel effects, use of guard-rings, latch-up, matching problems, substrate coupling, process gradients, and etching effect.
- Specifications for interface circuitry (voltage tolerances, level shifters, input loads, output drive strengths, buffers)

Also, design method issues such as the following must be resolved:

- Identify the overall technical risk areas.
- Propose and accept a simulation strategy.

Device-Level Physical Design

During this phase, the full schematic is transformed into a complete design layout. Because of the design-intensive nature of this phase and the need to consider many factors simultaneously, several iterations are often required. A critical factor for iteration is the block's sensitivity to layout parasitics and other non-linearities that are introduced during the layout. The focus here is to lay out all of the components and connections in the schematic as accurately as possible and to minimize parasitics on wires and components.

The first task is to create a layout template for the block that provides an overall size/aspect ratio derived from the top-level floorplan and a basic idea of where the devices should be placed, including their pin locations. Next, lay out the devices in the block. During this task, the devices must be generated and floorplanned. Power, guard rings, and device placement must also be planned. Finally, the devices need to be routed. Physical verification should be performed often at various stages of device layout to ensure DRC correctness and to reduce the number of iterations by catching errors early in the layout process.

Device-Level Floorplanning and Placement

A designer can begin by synthesizing (with tool assistance) or by instantiating (manually) devices/cells to generate all the needed instances, according to schematic instances. You will also need to generate the I/O pins and estimate the process boundary box.

With all the physical components created, a designer needs to do a rough component placement into the defined boundary box to verify that the provided boundary box is sufficient. This placement of the components must be consistent with the analog and digital partitioning (for noise immunity and performance) and should allow for the space needed for buses, nets, and guard rings.

The I/O pins should be placed or arranged within the boundary box. At this stage, the focus is to finalize the block geometry and place the pins in their intended locations. At the top-level of the physical design (assuming all routing metals are used), both digital and analog I/O pins must be placed around the periphery.

To properly align with other blocks in the design (system level), the cell boundary and pin locations must be fixed. With the pin locations and boundary box size satisfied, create or save an instance for this block. From the output of this step, a designer can now measure the core size and shape. The designer can use the measurement to generate an abstract cell that the system design level (a higher level design) needs to begin the design. This saves time and lets designers work in parallel. At the same time, designers must commit to the fixed I/O pins and die shape and size.

Running DRC checks to ensure that placement meets with the design criteria is a recommended step after the proposed placement is completed.

Device-Level Routing

Once placement has been completed, the next step is to hook up all the devices and components. A good floorplan and placement is key to achieving good routing results.

The routing layers to be used to route the block should be determined. For better routing traffic, metal layers are assigned for routing in either the vertical or horizontal direction. This helps to avoid shorts and to manage routing traffic. In most routing practices, odd numbered metal layers are assigned for horizontal routing, even numbered metal layers for vertical routing.

In digital blocks, higher-layer metals are typically reserved for top-level routing and are not available at the device-level. For analog/ mixed signal blocks, however, all available metal layers might be used because routing over the top of these blocks is generally avoided. Tracks

for feedthroughs should be reserved, if required. Provision should also be made for spare tracks.

Decide on a power routing strategy for the block. Often an existing power template (as dictated by top-down floorplan) is used for the purpose. Critical nets also need to be identified with non-minimum width, special spacing, shielding, and other such requirements in mind. These and any other electrical constraints that are defined in the top high level specification need to be implemented.

Once the routing has been completed, results should be analyzed and iterations made. Design rule checks should also be run on the routed block.

Physical Verification

Physical Verification ensures that the design layout meets the process requirements for the purpose of fabrication and implements the circuit correctly. It generally consists of the following checks:

- Design Rule Check (DRC)
- Netlist extraction (for ERC/LVS)
- Electrical Rule Check (ERC, Optional)
- Layout versus Schematic (LVS)
- Exclusive OR (Optional)

Parasitic Extraction and Backannotation

The purpose of this phase is

- To extract device components with resistance and capacitance parasitics in order to prepare the layout netlist for postlayout simulation
- To backannotate the schematic with parasitic information for the purpose of simulation or documentation

Postlayout Analysis

The outputs from the layout will be the extracted layout and the backannotated schematics, which can be used for postlayout simulation. These will be used to verify performance and to ensure reliability.

Engineering Change Order

If there are engineering design changes needed as a result of the analysis, an engineering change order (ECO) might be created to specify these changes. The physical layout tools have features to preserve the original design while making these changes. The layout designer will have to study the changes to determine their impact on the design. Minimizing the changes also minimizes the chances of having further problems with postlayout analysis.

Engineering change orders are also used to make design improvements after the product has been fully completed and shipped to customers. These same techniques can be used to minimize the impact of these changes.

Final Steps

Once the layout is complete, a General Data System (GDS) output needs to be generated to fabricate the chip (or for export to another environment). Once the design is translated and a test of the fabrication process has been completed, the design can be archived.

2

Design Environment

Data Preparation for the Automated Custom Physical Design (ACPD)

The design environment for the ACPD flow must be set up properly. This chapter discusses that setup. Refer to the Automated Custom Physical Design: Device Level Methodology Guide on page 93 for recommended practices of data preparation. You will need to set up the following:

- `.cdsenv` file
- .`cdsinit` file
- Display resources

Design Environment

Once DFII has been installed, the design environment needs to be configured to allow user preferences. This section describes some of the configurations and setups. Further information can be found in the *Cadence Design Framework II Configuration Guide*.

The .cdsenv file

In DFII you can customize your environment by creating or modifying an existing `.cdsenv` file. To create or edit an existing `.cdsenv` file, set the environment variables through the Display Options, Editor Options, and XL Options forms. For example, set the grid through the Display Options form, then create or modify an existing `.cdsenv` file with the Save Defaults form using the *CIW – Options – Save Defaults* command. The settings will be saved to the `.cdsenv` file.

A sample `.cdsenv` file is in *`your_install_dir/tools/dfII/samples`*. The `.cdsenv` file can be located in *`your_install_dir/tools/dfII/local`*, your home directory, or your current working file(s) will be read when DFII is first started.

3

Parameterized Cells

A parameterized cell (pcell) is a cellview that can be instantiated at run time based on different values of parameters associated with it. A special SKILL procedure is associated with the cellview that allows the dynamic creation of a pcell instance and modification of pcell parameters. The parameters are specified as a property list on the pcell cellview. This list lets you specify default values, and value types for the parameters.

When an instance of a pcell is placed into a cellview, all properties associated with the instance override the default parameters stored on the master.

Creating a pcell helps automate the creation of layout design data. Pcells should be designed as standalone entities, independent of the environment in which they are created and independent of the environments in which they are used.

A sample library of typical pcells is available. Refer to the *Sample Parameterized Cells Installation and Reference* for more information.

SKILL-Based Pcells Using Relative Object Design Constructs

The relative object design (ROD) SKILL constructs allow quick, intuitive, technology-independent cell generation. The ROD objects are similar in function to regular DFII layout shapes. The ROD objects allow many new concepts including:

- System and user defined handles, e.g. points on any figure's bounding box, as dynamic database objects
- Allowing the naming of shapes which makes possible the extraction of named objects (and handles) at any level of hierarchy using a named path. (i.e. "I0/24/polyRect")
- The ability to align figures using extracted point handles and technology file–based design rules.
- The ability to create a multi-part-path composed of multiple paths and rectangles.

- The ability to create hierarchical parameterized cells more easily.

Some of the techniques used in the ACPD flow require some advanced SKILL-based pcells with ROD constructs. There are examples of the SKILL-based pcells included in the regular install of the Virtuoso XL layout editor. Refer to the manuals for more information on pcells and CDF:

- *Sample Parameterized Cells Installation and Reference*
- *Component Description Format Users Guide*
- *Virtuoso Relative Object Design User Guide*
- *Custom Layout SKILL Functions Reference*

Calling SKILL Procedures from Pcells

User-defined SKILL procedures can be called from pcells. However, the called procedures must follow all of the rules applicable for SKILL routines for pcells. Called procedures are not compiled within the pcell, so they must be complied before the system evaluates the pcell. For information about how to attach code files to a library, see *"Advanced Features"* in the *Virtuoso Parameterized Cell Reference*.

What to Avoid When Creating Pcells

Here are some important guidelines for code used to create pcells:

- When creating a pcell, do not use functions with prefixes other than `db` and `tech`. Although it is possible to call procedures with other prefixes, such procedures will not be able to view the pcell in a different environment or translation required by another database. Most translators can translate only basic SKILL functions and functions in the SKILL families `db`, `dd`, `cdf`, and `rod`.
- Do not use functions with the prefixes `ge`, `hi`, `las`, `le`, `pr`, and `sch`. These functions could have functionality that might not work as required in a different environment.
- Do not prompt the user for input.
- Do not generate messages; message output is interpreted as an error.
- Do not load, read, or write to files in the UNIX file system.
- Do not run any external program that starts another process.
- Do not generate output with `printf`, `fprintf`, or `println`.

Being unable to print presents some problems for debugging. For other alternatives, refer to *"Advanced Features"* in the *Virtuoso Parameterized Cell Reference*.

4

DFII Library Development

Technology File

The Virtuoso® XL Layout Editor (Virtuoso XL) requires technology-specific information to be stored in a technology file for the design library. The following sections of the technology file must be completed in order to run Virtuoso XL.

- Virtuoso XL rules
- Layer rules
- Devices
- Physical rules

Virtuoso XL Layout Editor Rules

These rules are specific to Virtuoso XL. Virtuoso XL will not be able to start unless *lxExtractLayers* are defined in the technology file.

Extract Layers

lxRules specifies the layers the connectivity extractor will use for interconnect and vias. Specify a layer-purpose pair or the layer. If the layer is defined with no purpose, then all purposes are used.

Layers must be defined in *lxExtractLayers* of the technology file for the Virtuoso XL connectivity extractor to function.

Overlapping Layers

lxNoOverlapLayers specify layers that cannot overlap without receiving an error. For example, if poly and diffusion overlap in the design, an error marker will be placed at the

intersection of the two shapes. Layers defined in *lxNoOverlapLayers* of the technology file must also be defined in *lxExtractLayers* of the technology file.

lxBlockOverlapCheck

lxBlockOverlapCheck property, defined on a shape, instance, or instance master, tells the extractor not to check whether a non-overlap layer of this shape or instance is touching a non-overlap layer at the current cellview level or at different cellview levels.

lxBlockExtractCheck property, defined on a shape, tells the extractor not to check the connectivity between this shape and others it touches.

Layer Rules

The layer rules class of the technology file defines the rules between layers. These rules define layers that make up vias and define equivalent layers between layer-purpose pairs.

viaLayers

Layers that are used for vias must be defined in *viaLayer* of the *LayerRules* class of the technology file. If *viaLayers* are not defined, the flight lines will not update correctly when manually routing the design.

equivalentLayers

Define layers that have equivalent connectivity in *equivalentLayers* of the technology file. *equivalentLayers* define two different layers that are electrically equivalent without requiring a via or contact in between. Define both layers as *lxExtractLayers.*

Devices

Define all contact types in the technology file in *devices* class. Contacts are used by the *Create – Contact* and *Create – Path* commands. The connectivity extractor will propagate connectivity through symbolic contacts.

Symbolic Contacts

Symbolic contacts are used for making connections between different layers. The layers that make up a symbolic contact are defined in *viaLayers* of the technology file.

Physical Rules

Physical rules defines the minimum width of paths that are created for interconnect. The manufacturing grid is also set in the physical rules class.

Device Requirements

Properties added to devices include

- Permuting pins
- Multiplication factor
- Ignoring parameters
- Stop view list
- Specifying a cell from another library

Where to Add Properties

These properties can be added to any one of the following locations in the design library:

- The master symbol view
- An instance of the symbol in the schematic
- The master layout view
- An instance of the layout in the layout view
- The CDF for the device cell

Permuting Pins

Add *permuteRule* to a device to make the pins permutable.

`CDS_Netlisting_Mode` must be set in order for the pins to permute.

Multiplication Factor

Virtuoso XL supports use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout.

The m-factor can be a property on a device at the top-level or on a lower level of your design hierarchy.

You can implement the m-factor as a CDF parameter or as a property of an instance or a cell. As a property of an instance or a cell, it can be a string, an integer, or a floating-point number.

You can use an environment variable, `mfactorSplit`, to control whether the m-factor produces multiple layout devices or not.

You can also add a Boolean property, `lxMfactorSplit`, on a given instance to override the global value given by the `mfactorSplit` environment variable.

You cannot use m-factors with components that cannot be used in parallel, such as voltage sources.

Ignoring Parameters

Add *lxIgnoredParams* to ignore irrelevant parameterized cell (pcell) parameters in Virtuoso XL checking. For example, when the left contact or right contact parameter of abutting devices is turned off. When the layout is compared to the schematic, multiple warnings appear in the Virtuoso XL Info window:

```
‘rightCnt’ on source figure ‘P0’ is TRUE’
... on layout figure’|BUF_1|P0’ is ‘FALSE’
```

To suppress these warning messages when comparing the schematic to the layout, add the following parameter:

```
lxIgnoredParams =”rightCnt?”
```

Stop View List

Use *lxStopList* to identify the layout views that are used in layout generation.

- If you do not specify *lxStopList*, the default value is `layout`. If there is no layout view, layout generation will use the first view of *maskLayout* and gives you a detailed warning message in a Virtuoso XL Info window. A similar window appears if there are no views of *maskLayout* for the cell.

- *lxStopList* is a `string` which can be a list of views. The first view found is used during layout generation. After layout generation, you can change the view using *Virtuoso XL – Connectivity – Change Instance View*. The Change Instance View form displays the list defined in *lxStopList*.

5

Virtuoso XL Layout Editor

Connectivity Sources

The inherent power of the Automated Custom Physical Design (ACPD) is that the layout cellview has real-time connectivity extraction and instant visual feedback for violations. This will reduce the verification cycle in many cases to just one LVS pass. The connectivity source in the ACPD flow can be in the form of a netlist (CDL) or a Virtuoso® Composer schematic. Refer to the Automated Custom Physical Design: Device Level Methodology Guide on page 93 for recommended practices for device-level layout.

Schematics Source

If the connectivity source is a composer schematic, it must be in a valid format and able to be checked and saved. Ensure that lower-level schematics in the hierarchy can be traversed and all components that make up the schematic are in the current Library Manager path. Refer to the *Virtuoso Schematic Composer Tutorial* and the *Inherited Connections Flow Guide*.

Netlist Source

If the connectivity source is a CDL netlist that is copied or linked to a textual cellview, choose *File – Import – XL Netlist* to read in the netlist and generate connectivity in the layout. Virtuoso XL can use this directly to generate components and connectivity in the layout.

Placing Devices

There are three ways to place pins and devices in the layout design window.

- Choose *Design – Generate from Source* to place all pins and instances in the layout window.

- Choose the *Design – Generate from Source* command to place only the I/O pins. Then use *Create Pick from Schematic* to place the instances.
- Choose the *Edit – Place from Schematic* command to place the devices, as in the schematic, within the prBoundary.

Generate from Source

- Creates prBoundary
- Transistor chaining and folding
- Generates pins and instances

1. Select *Tools – Design Synthesis – Layout XL* (from a schematic).
2. In the Virtuoso XL Options form leave the option to *Open Existing* and click *OK*.

   Note: If you do not have a corresponding layout cellview, select *Create New*.
3. Click *OK* in the Open File form.
4. Select *Design – Gen From Source*.

   The Layout Generation Options form appears.
5. Turn on the *Instance* button.

   All instances will be placed in the design below the prBoundary.
6. Change the attributes of any pins.
7. Click *OK* to generate the design.

Pick from Schematic

- Specify instances and /or pins to be placed
- Select from schematic or listing
- Chain and fold instances

1. Select *Tools – Design Synthesis – Layout XL* (from a schematic).
2. In the Virtuoso XL Options form leave the option to *Open Existing* and click *OK*.

   Note: If you do not have a corresponding layout cellview, select *Create New*.
3. Click *OK* in the Open File form.

4. Select *Design – Gen From Source*.

   The Layout Generation Options form appears.

5. Turn off the *Instance* button.

   Only pins will be placed in the design.

6. Change the attributes of any pins.

7. Click *OK* to generate the design.

   You will use the Pick from Schematic form to place the instances in the layout.

8. Select *Create – Pick from Schematic*

9. Select one of the symbols in the schematic and move the cursor into the layout window.

10. Click anywhere inside the prBoundary to place the devices.

11. Select another symbol.

12. Select *Place Individually*.

13. The Pick from Schematic form updates. This option will allow you to place each of the instances from the corresponding symbol individually.

14. Move your cursor into the layout window and click to place the device.

Editing Placement

Listed below are the commands to edit the placement of the design.

- Move
- Transistor Chaining
- Transistor Folding
- Abutment
- Permute Pins
- Swap devices
- Align devices

Move Devices

Use *Edit – Move* to move devices and pins. The selected device is highlighted in both the schematic and layout. Drag lines are displayed from the pins on selected components. The drag lines are connected from an instance pin to the nearest instance pin on another device or to the nearest I/O pin. As the component is moved, the drag lines rubberband. Virtuoso XL does not show all possible connections, only the nearest one. The rubberbanding drag lines are only available when enabled on the Move form. When objects move, the net incomplete flight lines update. You can also use *Show Incomplete Nets* to display flight lines for incomplete nets to guide your move.

Transistor Chaining

Transistor chaining can be started/used from the *Generate from Source*, *Pick from Schematic*, or *Edit Transistor Chaining* commands. *lxComponentType* must be added for each type of device that is going to be chained. Add *lxComponentType* through *Design – Component Types*. The source/drains of devices are automatically shared by setting the *Transistor Chaining* option from the Layout Generation form. All devices that can share their source/drain will be abutted during layout generation. Devices can be chained individually by using the *Edit – Transistor Chaining* command. For transistor chaining to function correctly, make sure that *Auto Abutment* is turned on in the Virtuoso XL Options form.

Transistor Folding

Devices can be split into different width gates by using the *Edit – Transistor Folding* command. Specify the number of gates and the width of each gate. Split devices are automatically abutted. *lxComponentType* must also be added for each device type. Transistor folding can also be started from *Design – Generate from Source* and *Create – Pick from Schematic*.

Abutment

Abutment can be done while moving devices as long as the abutment parameters have been added to the parameterized cells (pcells). Overlap any pins that are on the same net. Listed below are the requirements for auto abutment.

- Both instances must be pcells set up for abutment or cells with abutment properties.
- Add `abutRules` to the pcell.
- Both instances can be from the same or different masters.

- Add `abutClass` to the pins of the different masters to indicate they are abuttable to each other.
- Assign automatic spacing properties to instances you do not want to have abutted.
- Add `vxlInstSpacingDir` to specify the spacing direction between instances.
- Add `vxlInstSpacingRule` to specify how far to space instances.
- Both instance pins must be connected to the same net, must be defined on shapes of the same layer, and must have opposite abutment directions.
- The shapes of the two pins must be identical, or one of them must be able to be completely contained within the other.
- During placement, the *Connectivity Extractor* and *Auto Abutment* options in the Virtuoso XL Options form must turned on.

Permute Pins

Occasionally you can improve the routing interconnections by swapping pins. To swap pins use the *Connectivity – Permute Pins* command. The Permutation Information window will display information about the pins. Open and choose *Connectivity – Show Incomplete Nets* before you swap pins, in order to see the pins swap. Pins to be permuted must belong to different nets and must first be defined as permutable terminals.

Virtuoso XL Layout Editor Options

Auto pin permutation and abutment are interactive features controlled by the user. Set the use of these options through *Options – Virtuoso XL*.

The *Auto Permute* option allows pins of a pcell device to be swapped the same time as a connection is made to that device. This feature works with any cell that has permutable pins defined.

When the *Auto Permute Pins* design options is active (the default setting) in the Virtuoso XL Options form, Virtuoso XL automatically permutes pins in a component if doing so corrects a short caused by routing in the following situations:

- The routing was done interactively
- When using auto abutment
- While moving or stretching a shape (or using any operation that changes connectivity)

The *Auto Abutment* option aligns the edge of the moved device to the edge of the stationary device while sharing diffusion if connectivity allows. Exact placement of the transistor is not necessary.

The *Auto Permute* and *Auto Abutment* options are data dependent and are designed as a function of the pcell. When both options are turned on devices with pins that overlap are automatically abutted and pins are swapped.

Swap Devices

To create better interconnect between components, swap a component's location with another component's location.

Start the *Edit – Other – Swap Components* command, and select two devices to be swapped. These devices are highlighted on the schematic (if the schematic is the connectivity reference).

- The symbols in the schematic do not swap location (the schematic is a connectivity reference), only the components in the layout are affected.
- Connections to the components are not moved after starting the *Swap Components* command. If the new component placements cause opens, using the *Connectivity – Show Incomplete Nets* command will highlight the connections that are incomplete.
- The *Swap Components* command does not allow you to swap components that are part of a satisfied constraint (unless the swap maintains the constraint).

Align Devices

Use the *Edit – Other – Align* command to align any object along an object's edge, origin, or center.

If you want Virtuoso XL to space objects evenly when they are aligned, turn on *Apply Separation* and specify a minimum separation distance in the *Separation* field.

Note: The *Align* command does not read spacing information in the technology file; therefore, shorts can be created with this command, When a short is created, Virtuoso XL displays a marker over the short.

Manual Routing

The designer has the option to prewire the critical nets (clocks, powers, matched nets) using the Virtuoso XL tool. Interconnect can be created with any shape, as long as the layer that the shape is drawn on is listed in *lxExtractLayers* of the technology file. Use the *Create – Path* command to create interconnect. Paths that are prerouted in Virtuoso XL will be translated into the Virtuoso custom router as protected nets. As the interconnect is created, the extractor will automatically assign net information to the paths or shapes used as interconnect.

Engineering Change Order

The engineering change order (ECO) subflow is supported within the Virtuoso XL environment, though only for schematic-based designs. The Virtuoso XL layout editor allows you to make incremental changes to the schematic after completion of the initial layout. These changes may include removal or addition of devices or rewiring of nets. Use the steps below for an ECO.

1. Make the required changes to the schematic.
2. Run schematic extraction (check).
3. Use *Connectivity – Update – Components and Nets* to update the layout with the new schematic connectivity.
4. Use *Connectivity – Update – Layout Parameters* to update the layout with the new schematic parameters.
5. Delete extra devices.
6. Move new devices and pins into place.
7. Delete wiring shapes associated with shorts.
8. Export to the Virtuoso custom router.
9. Reroute the design.
10. Import the new layout into DFII.
11. Verify the design.

Analyzing and Updating

Before running a comprehensive verification check on the layout design with the Assura Diva verification tool, check the design to compare design elements and their parameters in the schematic with those in the layout.

- Check against Source
- Update Components and Nets
- Update Parameters

Check Against Source

Use the *Connectivity – Check Against Source* command to check the layout against the schematic. If any devices are missing in the layout view, the corresponding symbol is highlighted in the schematic view. The Virtuoso XL Info window appears, listing the instance in the schematic that does not have a matched instance in the layout.

If any extra devices are in the layout view, a marker is added to the layout, with associated marker text indicating the error. Use the *Verify – Markers – Explain* command to get a description of the associated marker text.

Note: This command does not check for missing or extra I/O pins or instances with `lvsIgnore` set to `t`.

Note: The highlighting is not visible if you use a netlist as the connectivity reference.

Update Components and Nets

The *Connectivity – Update Components and Nets* command updates the layout with missing devices, missing I/O pins, markers on extra devices, and updated connectivity. Extra devices are those that have no corresponding device in the schematic, are not contacts, and are not marked with *lvsIgnore* or *ignore*. Virtuoso XL places devices in the layout with the same orientation as in the schematic. Existing devices are not moved.

If any pins are missing in the layout, the Create Missing Pins form appears. Once you have specified the size and layer to use, Virtuoso XL places new pins below the prBoundary.

The *Update Components and Nets* command updates the connectivity of all existing components in the layout to match the schematic connectivity. The connectivity extractor automatically updates the connectivity of any existing routing in the layout.

After the new devices have been added and the connectivity has been updated on the components, the connectivity extractor then indicates where any shorts occur because of the new connectivity. *Show Incomplete Nets* highlights where any opens now exist.

Note: This command does not update parameter values on devices in either the schematic or layout.

Update Parameters

If any of the parameters have changed in your layout or schematic, update the corresponding devices using the *Connectivity – Update – Layout Parameters* command or by using the *Connectivity – Update – Schematic Parameters* command. These commands update the selected device parameters in either the layout or schematic to match the corresponding view.

Virtuoso XL Layout Editor Tips

MultiPart Path

Use multipart paths (MPPs) to create contact arrays, guardrings, or shielded wires. MPPs can be created either through SKILL or using the *Create – MultiPart Path* command. The MPP created in this manner is a composite zero-level object that can be stretched, moved, and chopped.

rodAlign

Use the `rodAlign` SKILL construct any time that you require a bidirectional alignment between objects. The objects must be ROD objects.

rodNameShape

To turn a regular object into a ROD object, use the *rodNameShape* command. Refer to the *Virtuoso Relative Object Design User Guide* for more information.

Legacy Data

Legacy layout data can be used in Virtuoso XL. In order for the Virtuoso XL extractor to establish connectivity on the placed layouts, pin information must be added to the cell. If the

cells were drawn with layout versus schematic (LVS) text and no pins, generate pins using the *Create – Pins From Labels* command.

The *Pins From Labels* command allows you to create pins in the current cellview or inside selected cellviews. If you have multiple cells to process, turn on the *Selected Instances* option. The form will update with all the layers that contain text inside the selected instances. Choose the output pin layers, and choose the size for each label layer found.

When the pins have been generated in the cellviews, use Virtuoso XL as you would with any pcell. Virtuoso XL can start at any level in the design hierarchy. The cells can be mapped to the schematic symbols using *Update Device Correspondence* and placed in your new layout cellview.

6

Virtuoso Custom Placer

Overview

The Virtuoso® custom placer is an automated solution for placing components and cells in both block and cell-level designs. You control placement using topological and geometric constraints. You can enter constraints into the Constraint Manager from either the Constraint Manager or layout cellviews. All placement activity is initiated from within Virtuoso XL. Refer to the Automated Custom Physical Design: Device Level Methodology Guide on page 93 for recommended practices on placing devices in your design.

Setup for Placement

For information about setting up your environment for placement in Virtuoso XL refer to the following sections in the "Using the Virtuoso Custom Placer" chapter in the *Virtuoso XL Layout Editor User Guide*.

- Identifying the Placement Translation Rules
  - Specify the rules file to be used in the `.cdsenv` file.

    layoutXL.placement rulesFile string `myrules.dat`
  - This is the same rules file used to translate to the Virtuoso custom router
- Setting Virtuoso XL Environment Variables
  - Several placement environmental variables have equivalent variables to control the Virtuoso custom router. Placer variables always override their router equivalents during placement.
- Setting Variables for MOS chaining and stacking control
  - Define parameters to control how MOS devices are chained and stacked.
- Abutment

- If auto abutment is enabled in the Virtuoso XL options form, then abutment is automatically performed during placement. Abutment properties must be added to the devices. Automatic abutment is mandatory for chaining and folding to occur.

- Boundary
  - A prBoundary must exist in the layout in order to run the Virtuoso custom placer.
  - Identify the boundary layer purpose pair in the translation rules file.

Defining Component Types

You assign a component type to PMOS and NMOS devices from *Design – Component Type*. This is a way of tagging certain devices into groups. You define component types for two reasons:

- To identify PMOS and NMOS cells to enable chaining and folding.
- To identify groups of devices for row placement.

Note: Currently you do not need to assign component types for area based designs.

When assigning devices to component types:

- The *lxComponentType* is automatically assigned through *Design – Component Type*.
- The MOS parameterized cell (pcell) masters can be defined as any type; you need to set the *lxDeviceType* parameter for NMOS types to be *NMOS* and for PMOS to be *PMOS*. This indicates to the Virtuoso custom placer that all devices belonging to these types are MOS devices.
- The *lxActiveLayer* sets the spacing between the diffusion layers of the devices.

Important

Do not assign cells multiple component types. The Edit Component Type form currently lets you do this.

- Parameter *lxDeviceWidth* for device width needs to be defined for NMOS and PMOS devices, for example, the string may be `w` which should correspond to the parameter used in the pcell masters.
- Parameter *lxMaxWidth* needs to be defined for the NMOS and PMOS devices to specify the default width for chaining and folding to occur.

Important

Be consistent with the specified values. Virtuoso XL cannot accurately support a mix of values. For example: meters (10um is `1-e-5`) and user units (10um is `10`).

Pin Placement

The *Placer – Pin Placement* command allows you to assign pin constraints prior to placement by the Virtuoso custom placer. You can assign the following constraints from the Pin Placement form.

- Fixed constraints
- Align pins to any boundary edge
  - Supports rectilinear boundaries
- Assign specific pin ordering for a group of pins
- Place pin groups with any given spacing between each pair of pins that have been ordered
- Expand/collapse iterated bus pins (Q<7:0> = Q<7>, Q<6>)
- Handles multiple pins on the same net
- Reports pins and their assigned constraints

You must have pins placed in the layout window in order to view the pins in the Pin Placement form. If you do not have a prBoundary, all messages go to the command interpreter window (CIW) and the Pin Placement form will not appear. You cannot place a constraint on a pin that will reside outside of the prBoundary.

Placement Styles

There are four different placement styles.

- Component-assisted mode for standard cell rows
- Component-assisted mode for MOS rows
- User defined rows
- Area/mixed placement mode

Component-Assisted Mode for Standard Cell Rows

Component-Assisted Mode for standard cells gives you the opportunity to assess how much room you need for placement based on the variables you define. One of the advantages of this mode of placement is that it generates the rows for you based on user defined input. The limitations and assumptions for this mode are:

- ❑ Applies only to row-based with single height standard cells.
- ❑ Limited to one rectangular region
- ❑ Horizontal or vertical rows, not a mixture
- ❑ All rows contain the same component types
- ❑ Assumed standard cell definition is power over ground at R0 orientation
- ❑ Simple GPPG, PGGP, PG, GP patterns drive component orientation

1. Determine the placement style.
   - ❑ Minimum rows needed
   - ❑ Number of rows
   - ❑ Horizontal utilization
   - ❑ Vertical utilization
   - ❑ Row spacing
2. You can optionally specify an area in which rows will be generated by providing coordinates or drawing a rectangle in the layout. For example, in a mixed row-based and area based design you would want to restrict the region for standard cells to a subset of the entire placement area. If a region is not provided the entire prBoundary area is used.
3. Next select *Update Estimate*. Only component types of STDCELL will be evaluated. Run Layout Generation or *Update – Components and Nets*.
4. Evaluate horizontal or vertical utilization factors against routing resources.
5. Next setup for the row definition by defining *Supply Properties* and *Component Properties*.
6. Generate the row definitions which can be viewed in the *Show All Row Data* section of the Placement Style form.

   Note: The *Supply Properties* are not included in the row information shown in the *Show All Row Data*. The row definition still contains the supply properties, but it is not reported. If you are not sure that the definition is correct, redefine the rows.

7. Switch to *User Defined* mode and add or modify row parameters.

   Note: Once you manually alter rows generated with the *Assisted Mode* through additional layout generation steps, or modifying the region for placement, the design statistics previously displayed will no longer be valid.

Component- Assisted Mode for MOS Rows

Component-Assisted Mode for MOS rows gives you the opportunity to size up how much room you will need for MOS, based on the variables you define. The main advantage of this mode is that it generates the rows based on your input. The limitations and assumptions for this mode are:

- ❑ Applies only to row-based MOS placements
- ❑ Limited to one rectangular region
- ❑ Horizontal or vertical rows only
- ❑ Each row has unique MOS component type
- ❑ Simple NP, PN, NPPN or PNNP patterns
- ❑ All repeated rows have the same device alignment with respect to the rows (inside outside, or center).

1. Determine only one of the placement styles
   - ❑ Minimum rows needed
   - ❑ Number of rows
   - ❑ Horizontal utilization
   - ❑ Vertical utilization
   - ❑ Fixed row spacing

   Note: Because Assisted Mode runs layout generation, component type parameters *lxDeviceType* need to be set for PMOS and NMOS devices. The *lxDeviceWidth* and *lxMaxWidth* need to be set to evaluate folding parameters. The *lxActiveLayer* is needed for the Virtuoso custom placer to run connectivity.

2. Optionally specify an area in which rows will be generated by providing coordinates or drawing a rectangle in the layout. For example in a mixed row-based and area based design you would want to restrict the region for MOS to a subset of the entire placement area. If a region is not provided the entire prBoundary area is used.

Note: You may need to account for pins on the boundary by reducing the size of the rectangle.

3. Next select *Update Estimate* to run layout generation. Only component types of PMOS/NMOS will be evaluated and regenerated by layout generation given the following conditions.
   - ❑ Layout Generation has not been run yet.
   - ❑ Connectivity has changed
   - ❑ Device folding thresholds have changed
4. Evaluate the balance of P and N device counts against a placement style with equal numbers of P and N rows. Also consider horizontal or vertical utilization factors against routing resources.
5. Finalize setup for the row definition by defining *Supply Properties* and component properties.
6. If the desired results can not be achieved by varying folding thresholds, you can alter the region specified for assisted row generation.
7. Generate the row definitions which can be viewed in the *Show All Row Data* section of the Placement Style form.

   Note: The *Supply Properties* are not included in the row information shown in the *Show All Row Data*. The row definition still contains the supply properties, but it is not reported. If you are not sure that the definition is correct, redefine the rows.
8. Switch to *User-Defined* mode and add or modify row parameters.

   Note: Once you manually alter rows generated with the *Assisted Mode*, the design statistics previously displayed will no longer be valid.

User Defined Rows

The *User Defined* mode provides a full range of row properties for quick row definitions. Used by itself or as a way to customize rows automatically generated with one of the assist modes.The limitations and assumptions for this mode are:

- ❑ Unlike *Component-Assisted* mode there are no *Estimate* density statistics that guide the user as to how many rows will be needed.
- ❑ There are no limitations with regard to row definition. You can have any component alignment or multiple component types within a row.
- ❑ Supply properties cannot be added through the form.

1. Run a quick estimation using *Component Assisted* mode to help in determining your row area/configuration. This will determine how much area you need.
2. Enter user-defined names for the rows. This will be helpful in tracking which row is defined for what purpose.
3. Define row properties and component properties for the rows.
4. Select *Create New* to create the rows.
5. Select *OK* to write the row definitions to the cellview.
6. Run the *Auto Placer*.

Area/Mixed Placement Mode

Area based mode can be the least restrictive of any of the different placement modes. The Virtuoso custom placer is designed to place not just devices contained within rows or restricted to within fences but anywhere within the prBoundary.

The preferred method of placement that yields the best results is to confine devices to rows for device-level CMOS/standard cell, and define fences containing the remainder of devices. A generic fence definition containing all random logic not assigned to rows or highly constrained fences will provide the best results.

Placement is iterative. You can step through the entire placement sequence by first pre-placing and locking devices, defining rows and placing a group of devices. Next area place the rest of the design.

1. Assign constraints in the Constraint Manager first. Listed are the types of constraints the Virtuoso custom placer currently supports.
   - ❑ Distance
   - ❑ Alignment
   - ❑ Grouping
   - ❑ Symmetry
   - ❑ Fixed

   Refer to the online documentation for complete information on the constraint manager.

   Note: Constraints are treated differently by different tools. Constraints can be entered at the schematic level or during layout. Some constraints only make sense at the layout level. Constraints entered in the layout will be overwritten whenever layout generation is

rerun. Schematic constraints will be applied each time a layout is generated.

2. If applicable define rows with *Component-Assisted* mode.
3. If applicable define rows with *User-Defined* mode.
4. Run the *Auto Placer*.

Auto Placer

The Auto Placer places the devices in a specified cellview using the options from the Auto Placer form. Use the options from the Auto Placer form in combination.

- *Place Selected Objects* is a good way to partition up your design by using the *Edit – Search* command in Virtuoso XL to select only certain types of instances for placement.
- Turn *Allow Rotation* off to override defined allowed rotations and temporarily fix rotations for the placement run.
- *ECO Mode* is a standalone setting which is similar to a detailed placement pass. Only new devices will be placed. All other devices will not be disturbed, only shifted to make room for the new devices.
- *Global Placement* should always be run before any *Optimize Placement* passes.
- *Optimize Placement* has three settings, *Quick*, *Moderate*, and *Optimized*. The three settings have different degrees of wire length reduction and quality depending on the design style.
- The *Run Row Spacer* will compress or expand the rows in your design. The compression/expansion is between 0% and 10%. Zero percent would be just enough room for routing. 10% being an additional 10% routing resource. This is a generalized estimation, and does not take into account all routing resources.

Note: The Virtuoso custom placer will not change the size of the prBoundary.

- By default the technology file rule information is used by the Virtuoso custom placer. If the Virtuoso custom placer does not find (all of the necessary the Virtuoso custom router rules information) then specify translation rules file.
- *Filler Cells* is for standard cell designs. Filler cells will be inserted to fill all the gaps between standard cells.
- *Save As* redirects the results to another cellview. This is good practice when first getting used to the Virtuoso custom placer. If *Save As* is turned off the results will automatically get imported to your current cellview, not to disk.

Basic Placement Flow

The flow of placement-related tasks is very flexible, and many of the steps below can occur in different orders, or be repeated. Placement is an iterative process, and the steps below are meant to describe only the most generic flow.

1. Select *Design – Gen From Source* and generate pins, instances, and boundary.

   You could also create the prBoundary from *Design – Gen From Source*, and then use *Create – Pick from Schematic* to place the pins and instances. Pins can also be created from the *Create – Pin* command.

2. Constrain pins using the *Place – Pin Placement* command.

3. Set placement mode by invoking *Place – Placement Style*.

   Assisted Mode runs *Gen from Source* for MOS only. Therefore the design will be regenerated. User defined mode preserves pins from the schematic.

4. Add constraints to the layout by invoking *Tools – Constraint Manager.*

5. Run the Virtuoso custom placer with the *Place – Placer* command.

6. Update constraints as needed and run *Place – Placer* again.

Your own design flow will vary depending on the type of design you need to place.

Constraint Manager Support for Placement Constraints

Placement constraints let you control the Virtuoso custom placer. There are a variety of constraints, which you set using several different parts of the Virtuoso XL user interface, including The Constraint Manager, which you open with the *Tools – Constraint Manager* command in the CIW.

You can constrain any device in the layout at any time, except while the Virtuoso custom placer is running. Typically you would use the Virtuoso custom placer iteratively, setting more and more strict constraints as you refine the placement.

Placement Constraint Types

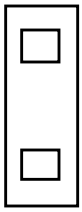
The Virtuoso custom placer recognizes constraints specified for:

- Components
  Component constraints determine device positioning, either relative or independent of each other in the layout.
- Pins
  Pin constraints assign pins to a particular edge, and optionally order the pins along each edge according to your specification.

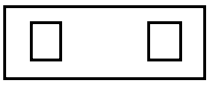
Fixed Position Constraints

Fixed constraints specify the X and Y location and orientation of an instance, pin, or polygon.

Origin fixed at X=10, Y=8



Horizontal orientation



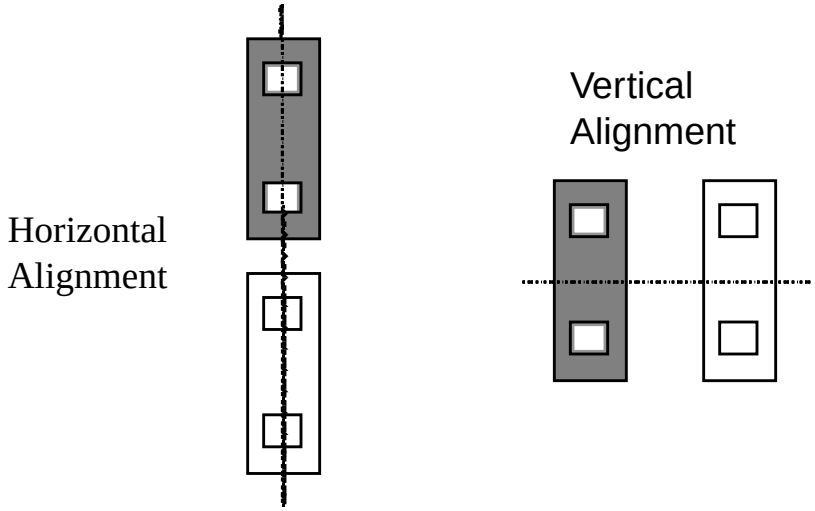
Fixed constraints on instances are only respected if the instances are located at their constrained position in the preplaced view. The Virtuoso custom placer will not move instances to their constrained position.

Alignment Constraints

Alignment constraints align two or more objects (instances, pins, or rectangles) by

- Their bounding box centers
- Any bounding box edge

- Any edge of the bounding box of any layer in the object



You can also specify the relative order of the aligned components: left to right or right to left for horizontal alignments and bottom to top or top to bottom for vertical alignments.

Alignment constraints on instances are respected if they are aligned in the preplaced view. However, if they are not aligned in the preplaced view, the Virtuoso custom placer does not move them into alignment.

Distance Constraints

Distance or separation constraints let you define minimum and maximum values for the distance in the x or y directions between two components (instances or pins). Distances can be measured between bounding box centers, specific edges, or the nearest edges, relative to the bounding box or a specified layer.

The Virtuoso custom placer tries to respect distance constraints but relaxes them if necessary to achieve a good placement solution.

Confinement Constraints

A confinement constraint restricts a group of components to a rectangular region (*fence*). You can optionally exclude all other, or specified, components from the fence.

The Virtuoso custom placer moves instances as necessary to implement confinement constraints even if they are not properly positioned in the preplaced view.

Hard Grouping Constraints

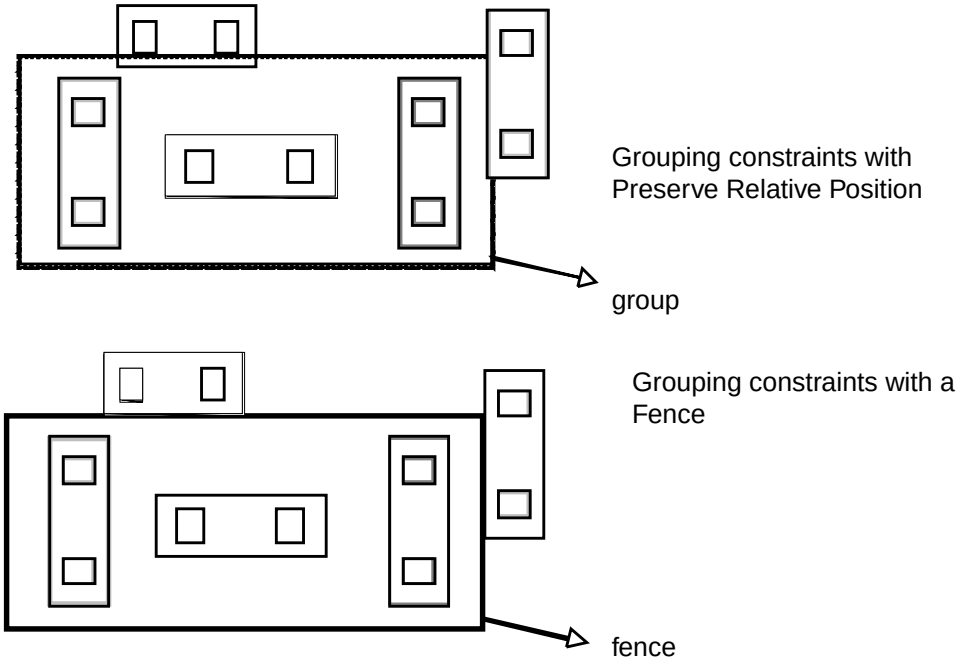
A hard grouping constraint (also called a firm group or hard fence) preserves the relative position and spacing of a group of components.

The Virtuoso custom placer moves instances as necessary to implement hard group constraints even if they are not properly positioned in the preplaced view.

Soft Grouping Constraints

A soft grouping constraint keeps a group of components close together somewhere within the cell boundary, but does not constrain the group itself to a particular location.

The Virtuoso custom placer moves instances as necessary to implement soft group constraints even if they are not properly positioned in the preplaced view.

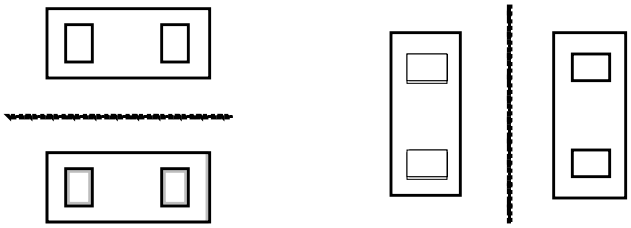


Symmetry Constraints

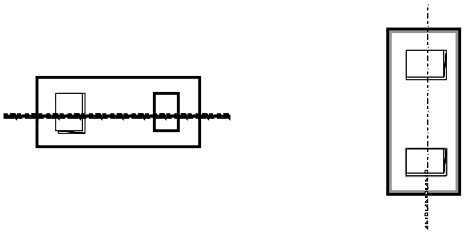
Symmetry constraints require a set of components and component pairs (instances and pins) to preserve their symmetry with respect to an axis. The axis can be either vertical or horizontal, and its position can be either fixed or variable.

Note: You can create symmetry constraints with the Constraint Manager, but the Virtuoso custom placer does not recognize them.

Symmetry pairs



Self-symmetric objects



7

Translation

The Virtuoso® layout data must be extracted into the native Virtuoso custom router design file format. The files are generated and written to disk for the Virtuoso custom router to read on startup. The translation rules define all the shape manipulation in an easy-to-use form-based interface.

Message Passing System (MPS)

The Message Passing System (MPS) is the pipeline for sending information between Design Framework II (DFII) and the Virtuoso custom router. This mechanism enables dynamic instance updates from Virtuoso XL to the Virtuoso custom router. Advanced users can also send SKILL commands from Virtuoso XL to the Virtuoso custom router.

- Change the parameters of a parameterized cell (pcell) in Virtuoso XL.
- Change the parameters of a pcell in the Virtuoso Composer window.
  - First generate a layout for the modified pcell with the *Connectivity – Update – Components and Nets* command.
- Edit the properties of any instance and change the library, cell, or view name.

Design Framework II (DFII) to Virtuoso Custom Router Translation

When DF files are translated to Virtuoso Custom Router the following occurs:

- Equivalent layers are merged with their master layer
- Pin shapes are identified
- For each layer, all shapes on that layer are merged together
- The merged shapes are sorted into conductors and keepouts

- Any shape that touches a pin, or touches a shape that touches a pin (traversing through vias), is a conductor
- Any shape that is not a conductor or a pin is a keepout
- Pin shapes are cut out of the conductor shapes
- The keepouts identified as pin shapes are merged with the user-defined keepouts.
- Pin shapes are oversized by the value of *minSpacing* and then cut out of the merged keepout shapes. Optionally, pins are voided to the edge of the keepouts in their access direction.
- DF via and contact cells identified in the translation rules are translated either as *via_images* or *via_array_templates* in Virtuoso Custom Router.

Translation Rules Editor

Once you are ready to start routing, you will need to export your Virtuoso XL layout data to the Virtuoso custom router. A rules file is necessary for translating the Virtuoso XL layout data to the Virtuoso custom router. Follow the steps for creating a new rules file.

1. To create rules for your process, select *Route – New Rules* in the layout window.
2. Select the library from which you want to translate the layout.

   The form that appears is empty.
3. Notice the radio buttons at the top labeled *Layers, Vias*. Select the *Vias* button to define the routing vias first.

   This will seed the Layers page of the Rules Editor with the layers used in the vias, saving you time when you set up the layer rules.
4. Select the appropriate vias for your technology, for example `M1_M2` symbolic and `M2_M3` symbolic.
5. Once you have at least two vias defined for routing, select the *Layers* button.

   Notice that layer information has been added for the layers in the vias.
6. Set the Palette Order column to make sure that the layer order is correct, ascending or descending. Generally, the lowest level of your mask set belongs on the bottom. If you are defining diffusion layers and polysilicon layers, make sure you order the diffusion layers after the polysilicon layers and that the *via* falls before the metal layer. You must also specify the connectivity of layers by specifying a Palette Order. The Palette Order is

how the Virtuoso custom router establishes connectivity for connecting layers by contacts and vias

7. Set the Layer function for each layer. Notice that the Layer function for `poly` is set to *Polysilicon*, and not *Metal*. This is necessary if you want pins on the same net that are connected by poly to be translated as identical pins of type *weak* when translating with the *Full Connectivity* option turned on. This is necessary to prevent *poly* gates from being used as feedthrus in the Virtuoso custom router.

8. You must specify all layers needed to route the design correctly for the Virtuoso custom router. Layers can be either set to *T* (translated) layers and/or *Ref* (referenced) layers. For device-level designs, it is typical to translate all device layers (`pdiff`, `ndiff`, `pwell`) for viewing-only as reference layers. Layers specified as Reference layers will be translated without connectivity. You can set Routing information such as direction, width, and spacing from this form.

9. Select the *Keepout* button. The Keepout page allows boolean generation of routing blockages to be defined for each layer.

10. Define keepout regions. For example, you need to show all metal1 but need to merge non-routable regions into larger keepouts to reduce the amount of data the Virtuoso custom router has to load into memory. You will need to oversize then undersize the data. The idea is to close any area that is not large enough to allow a minimum size route to pass through.

    Size = ((2 * minimum space) + (minimum width – grid)) / 2

    If metal1 minimum space is .6, metal1 minimum width is 1.2, grid is 0.1

    Size = (((2 * .6) + (1.2 – 0.1)) / 2)

    Size = 1.15

11. The translation rules to generate the desired keepouts are:

    metal1(drawing) SIZE 1.15 tmp_M1

    tmp_M1 SIZE –1.15 metal1(drawing)

    The temporary layer input box is the *Name* box in the middle right of the menu.

    If you need to do *black box* abstracts, make sure you have a different layer for the route boundary and the cell boundary (that is, prBoundary(drawing) vs. prBoundary(boundary)).

12. Define the cell boundary as the routing layer keepout using these rules:

    prboundary(drawing) => metal1(drawing)

prboundary(drawing) => metal2(drawing)

prboundary(drawing) => metal3(drawing)

13. If you need to create a 5um larger spacing rule around existing Metal 3 power supplies (halo), you will need to create a halo keepout. If metal3 minimum width is.8um and the required halo is 5um:

    metal3(drawing) SIZE 5.0 tmp1_m3

    metal3(drawing) SIZE 0.8 tmp2_m3

    tmp1_m3 NOT tmp2_m3 metal3(drawing)

14. Conductor translation rules for CMOS designs should identify the MOS conductor shapes. The Virtuoso custom router needs this information to properly handle routing to the device pins. Use the Conductors page of the Rules Editor to specify these rules.

    poly (drawing) C-NOT ndiff (drawing) = poly (drawing)

    Another workaround is to translate the data using Conductor Depth 0, Keepout Depth 32. This will write the design files without conductors thus defining everything a keepout. Then the weak connect pins will route correctly.

Translation Tips

Update Placement from Virtuoso XL Layout Editor to Virtuoso Custom Router

Here is an example of a custom SKILL routine to send information to the Virtuoso custom router.

Currently the placement information from Virtuoso XL is not sent to the Virtuoso custom router when the dynamic instance update takes place. The workaround is to select the instances and send the Virtuoso custom router the placement information via MPS. The following SKILL function will parse the instance properties and generate the correct the Virtuoso custom router command syntax. The last command is to set a bindkey for this function.

```
procedure(acpdUpdatePlacement()
let( ()
  compositeCmd = "checkmode off; "
  foreach(component geGetSelSet()
    if(component~>objType == "inst" then
```

```
      name = component~>name
      X = car(component~>xy)
      Y = cadr(component~>xy)
      rotation = cadr(component~>transform)
      case( rotation
        ("R0"   rotation = "0")
        ("R90"    rotation = "90")
        ("R180"   rotation = "180")
        ("R270"   rotation = "270")
        ("MY"   rotation = "0  (mirror y)")
        ("MYR90"  rotation = "90 (mirror y)") ("MX"   rotation = "0  (mirror x)")
        ("MXR90"  rotation = "90 (mirror x)")
      ); case
      sprintf(placeCmd "place %s %f %f front %s; "
          name X Y rotation )
      compositeCmd = strcat( compositeCmd "" placeCmd)
    ); if
  ); foreach
  compositeCmd = strcat( compositeCmd "checkmode on;")
  printf("ICC Command: %s\n" compositeCmd)
  iccSendCommand(geGetEditRep() compositeCmd)
) ; let
)
hiSetBindKey("Layout"
"<Key>F10"
"acpdUpdatePlacement()")
```

You can add this or any other user-defined function to the Virtuoso XL menus. For more information, see the following README file:

install_dir/tools/dfII/etc/tools/menus/README

TIP: Update Polygon Data from Virtuoso XL Layout Editor to the Virtuoso Custom Router

The Virtuoso XL polygon data is not dynamically updated. The only way to get the polygon data into the Virtuoso custom router is to re-export the design to a temporary location. You can achieve this without shutting down the Virtuoso custom router. In the Virtuoso custom router window, save a session file and then import the current Virtuoso custom router data into the Virtuoso XL layout cellview before you re-export. The new wires file will contain all the current Virtuoso custom router data and the Virtuoso custom router internal extractor will delete redundant wires.

The re-translated Virtuoso custom router wires file (filename.wir) can be read into the Virtuoso custom router and the new polygon shapes will appear.

You can add the connectivity information automatically in Virtuoso XL if it is connected to a wire/pin or change it manually in the Virtuoso custom router.

8

Virtuoso Custom Router

The Virtuoso® custom router provides the routing solution for the Automated Custom Physical Design (ACPD) flow.

The methodology described here is a suggestion to better guide you through a layout design flow. This is a generic flow. This approach is only one of many ways to achieve a handcrafted design in an automated environment. Refer to the Automated Custom Physical Design: Device Level Methodology Guide on page 93 for recommended practices on routing your design.

Router Overview

The Virtuoso custom router is a multi-pass, rip-up and re-route, re-entrant router. The multi-pass aspect of the Virtuoso custom router lets the routing algorithm change variables and focus on different aspects of the design. During each successive pass, the Virtuoso custom router has a new set of parameters and a new partially routed problem to complete. The rip-up and re-route aspect allows the Virtuoso custom router to leave violations during a route pass and come back and try to resolve them in successive passes. The re-entrant aspect lets you stop the Virtuoso custom router at any point during the autoroute session, make modifications by hand and then start the Virtuoso custom router again.

The Virtuoso custom router lets you set many controls to help achieve a good routing solution. The *Router Rule* menu is set in order of precedence with the items at the bottom of the menu having the greatest rule precedence.

Many of the Virtuoso custom router features are targeted for chip assembly, and may be unnecessary when working at the device-level.

The key to success with the Virtuoso custom router is iteration and experimentation. Many users find that quickly iterating through the design many times will result in a design close to or better than if the design were routed by hand. If you get frustrated with the design progress, stop to review the documentation and analyze your data.

Virtuoso Custom Router Files

All of the design files used by the Virtuoso custom router are UNIX formatted ASCII files.

Design file (*filename*.dsn) - The Virtuoso translated design contains six sections in six or seven files.

1. **IC –** This is the main section. This section describes the design name, library, path, version, delimiters. This section references the section below.
2. **Parser** – This section describes layer definitions, boundary, via images, grid, and some global rules. (*filename*.str)
3. **Placement** – This section describes the instance placement information. (*filename*.pla)
4. **Image** – This section describes each image (cell), via_image, and via_array_template. (*filename*.lib)
5. **Network –** This section describes the netlist pin-to-pin connections. (*filename*.net)
6. **Wiring** – This section describes any pre-routes and wiring polygons if they exist. This is an optional section. (*filename*.wir)

Color file (*filename*.color)- this user created file allows the color to be mapped from on design to another. For example:

```
background black, metal1 red, metal2 blue.
```

Wires file (*filename*.w) – This is a user saved wires file. It contains only wire and via information that the user has chosen to include. You can save different wire files and read them in at any time. The user can choose which nets/wires to write to the wire file. The net information in this file is comments only.

Route file (*filename*.rte) – This is a user saved file. This file contains the wires, guides, global routing path, vias and testpoints. It also includes protected and unprotected wires. The user can choose which nets/wires to write to the route file.

Placement file (*filename*.plc) – The placement file describes the individual image placements in the design.

Session file (*filename*.ses) – The session file saves all routes, placement, and some rules for the design. This file is imported into Virtuoso XL after routing in the Virtuoso custom router is complete. The session file can be loaded in the Virtuoso custom router like a design (*.dsn) file. If you are going to load the session file into the Virtuoso custom router, make sure you have the original design files that were used to generate the session file. There are references to the original design files at the top of the session file.

.do File (*filename*.do) – This is a list of commands that will be sent to the Virtuoso custom router in sequence.

.did file (datecode.did) – This file captures all commands that are used in the routing session. All of the commands in this file are legal Virtuoso custom router commands. This file can be used to generate a .do file.

Filenames used in a Virtuoso custom router command (entered in the Virtuoso custom router or used in a .do file) can be preceded by the prefix $/ for UNIX files or $\ for Windows files as a shorthand way to denote the design directory path. For example, to save the current routed wires in a wires file:

On a UNIX platform, enter

```
write wire $/controller.w
```

If the design directory is

```
/home/user/test
```

the file is written as

```
/home/user/test/controller.w.
```

On a Windows platform, enter

```
write wire $\controller.w
```

If the design directory is \home\user\test, the file is written as

```
\home\user\test\controller.w.
```

In addition, you can use the following variables in the Virtuoso custom router commands to denote the design file base name, filename, and title:

- *$dsn*, which denotes the base name of the design file
- *$designFile*, which denotes the filename of the design file
- *$designID* which denotes the design identifier that follows the IC keyword at the beginning of the design file.

For example, suppose the design file filename is controller.dsn and the file begins with the title:

```
(IC control_block
```

In this example, if you use the variables to specify filenames:

- *$dsn*.rte denotes the filename controller.rte
- *$designFile*.rte denotes controller.dsn.rte

- *$designID*`.rte` denotes `control_block.rte`

Command Entry

There are four ways to enter Virtuoso custom router commands:

- The graphical user interface (GUI) menus
- The *Command Entry* area, using the keyboard
- `.do` files
- Bindkeys

The preferred method for controlling the Virtuoso custom router is with a `.do` file, which is a text file that contains one or more Virtuoso custom router commands. Commands are executed in the order you enter them from the beginning of the file to the end. The purpose of the `.do` file is to record rules and other commands you use during an autorouting session. If you need to revisit a finished design, you can edit the original `.do` file and reuse it.

The commands can be stacked in the command entry window or in the `.do` file with a semicolon separating the commands. If you copy and paste the commands from a UNIX terminal window, the linefeed/carriage return is replaced with a semicolon automatically.

The Virtuoso custom router has a built-in rule `.did` file editor to record the commands that are generated by the GUI. The `.do` files generated this way are syntactically correct. The rules `.did` file editor menu is located at *Edit – Rules Did File*. This window can be used with the copy and paste commands with other text editors. Also you can cut and paste from the transcript window to generate a `.do` file.

Interactive Editing

You can interactively edit wires and polygons in the Virtuoso custom router. You can add wires in *Edit Mode* by clicking a pin, an existing wire, or polygon.

The polygon entry commands (*fence*, *boundary*, *wiring_polygon*, *region*, *keepout*) require a follow-up command to define the properties. After you have completed digitizing the rectangle or polygon, press the right mouse button and choose *Define Polygon as*. This command prompts you for specific properties for the new polygon. To digitize a rectangle, click and hold the left mouse button. To digitize a polygon, click on each vertex in the polygon. To close the polygon, click on the first vertex or click the right mouse button and choose *Define Polygon as*.

If you adding a *wiring_polygon*, you will need to add the net connectivity to the object once you have finished digitizing. Use the right mouse button *Change Polygon Mode* command to change the net and layer.

Layer Panel

The Layer Panel provides the control and visual feedback of the routing layer status and other system layer controls. It provides the ability to set the preferred direction (horizontal | vertical | orthogonal | off), current layer (pencil), set selection (S), and visibility (colored square). The four header items are buttons toggling all layers on or off for the respective function.

The system layers have selectivity and visibility controls. Some of the system layers, such as Guides, have setup information access with the button with containing three dots.

The Color Palette is where you make changes to the layer color and fill pattern information. Shown below the layer `poly` is set for vertical routing, is the current layer, and is selectable and visible

Layers Panel

Objects Types

The Virtuoso custom router has several different types of objects. The terminology in the router GUI and documentation differs from the terminology of other tools in the Virtuoso layout environment.

Image – An image is the cell description. This object is defined once in the library section of the design file and can be referenced for the individual component placements. This is the same as a Virtuoso cellview.

Component – The individual image placements. This is the same as a Virtuoso instance.

Area – An area is a shape that can receive route spacing rules. The valid shapes include keepouts, regions, and design boundaries.

Pin – A pin is the object that drives connectivity in the Virtuoso custom router. A pin is basically the same as in the Virtuoso tool.

Via – This refers to the objects that create the connection between interconnect layers. This term may be used to describe a contact between polysilicon and metal1.

Wire – This is a routing object. Wires can be any valid routing layer including polysilicon and diffusion. To interactively add a wire, use the top icon in the bottom section of the route-mode menu bar. The icon looks like an X with a tail extending left and to the bottom.

Keepout - A shape that the blocks the autorouter and interactive router from inserting objects (wires, via, polygons, bends) in a given area.

Attaching Rules to Objects

The Virtuoso custom router lets you attach rules to objects. These rules follow a strict hierarchy of precedence based on the order in which they appear on the pull-down menu. Items at the bottom of the menu take precedence over items at the top.

The rules submenus are very similar in format. Each submenu contains the same menu entries as other submenus in that family. For instance, *Class Layer***,** *Group Set Layer*, and *Net Layer* all have the same submenus.

Rules Submenu

The items in the bottom section of the menu control general routing and data checking.

You can tear off the menu and keep it on your desktop by clicking the dashed line at the top of the menu.

The example below shows how to use the *Rules – Layer – Clearance* command to attach clearance rules. From the menu you can adjust all of the parameters that define clearance rules for pin to pin, wire to wire, area to area, via to via, or any other combination.

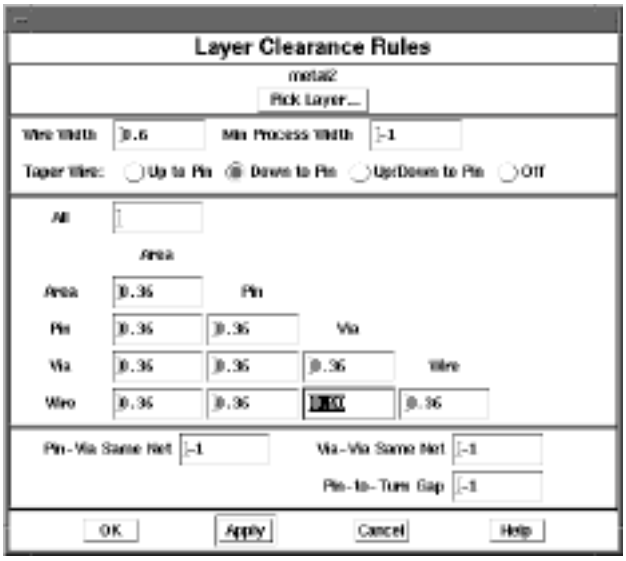
The matrix shows a relationship between each object type and all the other object types. You can set very specific rules, such as: metal2 space is 0.36 unless the length is less than 0.76 (the width of a Metal2 via), in which case the rule becomes 0.20. This is very typical for deep sub-micron technology. You would set the following rule using this command:

```
rule layer metal2 (clearance 0.20 (type via_wire))
```

Using the GUI:

1. Select *Rules – Layer – Clearance*.

The Layer Clearance Rules form appears.



2. In the Wire-Via cell of the array in the middle of the form, enter the value `0.20`.

Pin Types

There are three types of pin models in the Virtuoso custom router. This pin model describes the identical pin relationship. This default is set at translation time on the *Export to Router* form. These three models are similar to most other routing tools.

- Must join – tells the Virtuoso custom router to connect all identical pins on each net.
- Weak Connect – tells the Virtuoso custom router to connect only one of the identical pins on each net. (However, if they are already connected, the Virtuoso custom router might use a pair of weakly connected pins as a feedthrough.)
- Strong Connect – tells the Virtuoso custom router to connect at least one of the identical pins on each net.

Cost and Tax

Every object in the Virtuoso custom router has a cost. From one routing pass to the next, the Virtuoso custom router changes the internal cost of some objects. For example, when the Virtuoso custom router chooses to add a via, there is a cost associated with that via. Every section of wire, every bend, every via, every t-junction, every violation (LVS and DRC) – all of these objects have associated costs. Wires have many costs associated with them, all of which are summed to calculate the total cost of the net. For example, if routing layer `metal2`

has a preferred direction of vertical, the cost of routing horizontally would be higher than the vertical cost.

In the Virtuoso custom router, some of the costs associated with routing can be modified with the *tax* and *cost* commands.

The cost value changes from one routing pass to the next. If you enter a *cost* command, your value overrides the routing algorithm and prevents the cost from changing from pass to pass.

To keep the full benefit of letting the Virtuoso custom router to adjust costs, but still have influence on the cost values, use the *tax* command to enter a cost multiplier. Any floating-point number greater than 1.0 will increase the internal cost associated with that particular tax command. Any floating-point number less than 1.0 will decrease the internal cost.

Only use the *cost* command if you have exhausted all other means of controlling the Virtuoso custom router or you are certain the outcome will be positive.

The cost menus are:

- *Rules – Layer – Costing*
- *Rules – Costs*

Below are some syntax examples:

```
cost way forbidden
cost via low
cost layer poly forbidden
tax layer metal1 5.0
tax way 5.0
tax via 10.0
```

As the Virtuoso custom router progresses through the route passes, it narrows the amount of data it modifies. The table shows the narrowing scope of data from pass to pass.

Table 8-1 Narrowing of Router Data Scope from Pass to Pass

Route Pass	Routed Data
1-5	All nets ripped up and re-routed. Source/Terminator cycled.
6-15	Unroutes and nets with conflicts ripped up and rerouted. Source/Terminator cycled.
16-100	Unroutes and from-To with conflicts ripped up and rerouted.

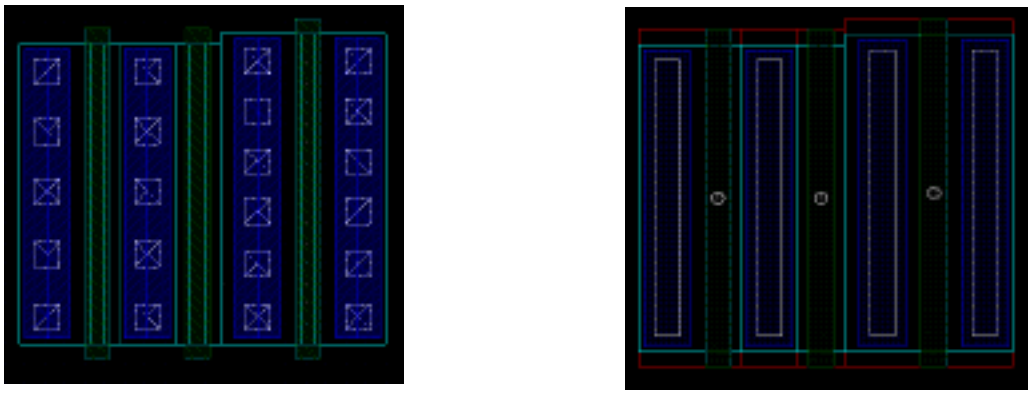
This, combined with the changing costs, gives the Virtuoso custom router a new, partially completed problem on every pass.

Transistor Modeling

The Virtuoso custom router transistor model is somewhat different from the Virtuoso model, because the extraction commands create a poly pin at the top and bottom of the transistor. Depending on the other extraction rules defined, there may also be a keepout shape over the gate region, diffusion modeled as poly, no contacts in source and drain area, or oversized contact keepouts.

The example below shows three parallel MOS devices in Virtuoso XL on the left. On the right are the same three MOS devices in the Virtuoso custom router.

Figure 8-1 Same Transistors in Virtuoso XL and the Virtuoso Custom Router



Notice that the contacts have been merged into long via keepout shapes. This keeps the Virtuoso custom router from dropping a M3_M2 via on the source/drain regions. The poly has been extracted as two weak connected pins with a poly conductor in between.

Data Integrity and Design Verification

The Virtuoso custom router rule checker differs from the typical design rule check (DRC), electrical rule check (ERC), and layout versus schematic (LVS) verification tools. The checker looks at very specific object rules during the autorouting process.

DRC/ERC/LVS extraction is always running unless you manually turn it off with the *Checking* button at the bottom left corner of the edit window. The button will turn red as a warning that checking is turned off.

The Virtuoso XL layout data may have blocked pins, errant keepouts, DRC/ERC/LVS errors, and similar problems, which may not show up until the data is loaded into the Virtuoso custom router. Always check all rules as soon as you load the design for the first time. If the Startup Warnings window opens upon loading the design, stop and fix the problems before you proceed with routing. Usually these warnings are pins that are shorted, undefined, or inaccessible in some way.

Pin Checking After Translation

The Virtuoso custom router includes many different online verification and data integrity checks, including a pin check for Virtuoso custom router access and physical geometry problems associated with pins. The command for this check is:

- *Rules – Check Pin Data*

  ```
  check(include pin)(exclude min_width_pin)
  ```

Once you have identified the pins in conflict, there are several ways to fix the problems.

- Go back to Virtuoso XL, make modifications, and then re-export the data to the Virtuoso custom router.
- Use `cut_keepout` on the chip, which automatically cuts top-level routing layer keepouts to provide access to pins. Top-level keepouts modified with this command will be saved in the session file and imported by Virtuoso XL. The command to do this is:
  - *Route – Preroute – Trim Keepouts – On Chip*

    ```
    cut_keepout 12 all cut_type chip)
    ```
- Use `cut_keepout` on images, which cuts keepouts inside images, to the nearest edge of the image. This is a virtual modification to the image and only stored for the current routing session. These changes will not be saved to disk. The command to do this is:
  - *Route – Preroute – Trim Keepouts – On Images*

    ```
    cut_keepout to_edge (cut_type image)
    ```

Design Rule Check (DRC) and Routing Checks

The DRC and routing checks can be run at any time. They are always on during autorouting. You should run them manually if you are routing the design interactively. Working interactively, you can set new rules on an existing route and test multiple scenarios. For example, you could check the effect of setting a crosstalk rule on a previously routed design.

Figure 8-2 Setup Check Rules Menu

Users often turn off same-net checking during the autoroute passes and turn it on for the clean passes. Same-net checking makes the Virtuoso custom router run more slowly because it has to check all of the nets against themselves.

The protected pre-route wires that come in from Virtuoso XL will not be checked because protected-wire checking is turned off by default.

The command to control the DRC and routing checks is:

- *Rules – Check Rules – Setup*

Below are some examples of the check command:

```
setup_check (conflict off)
setup_check (xtalk on)
check(type route)
check(type all)
```

There is also a *check_area* command on the vertical icon bar. This command only checks the current window area using the setup from this menu.

Pre-Routing Critical Nets

The interactive editing capabilities in the Virtuoso custom router and the Virtuoso XL tool may provide a better solution than the auto router. In this case, you can pre-route some nets in the editor of your choice and then autoroute the remaining nets.

To interactively add a wire, use the top icon in the bottom section of the Route-Mode Icon Bar. The icon looks like an **X** with a tail extending left and to the bottom. To add a via and change to a new layer, double click the Right Mouse Button or Space Bar.

You can select a set of nets, autoroute them with one set of routing commands, and then protect the wires. Then you can redefine the routing setup for the next set of wires. Protected wires are not modified by subsequent route passes. Your protected nets might include power supplies and other critical nets.

Be careful not to block the Virtuoso custom router access to other pins.

You can probe from the schematic or layout cellview to see the critical net in both Virtuoso XL and the Virtuoso custom router.

Using Fences

Virtuoso custom router fences provides a pseudo-boundary inside your design. The hard fence forces the Virtuoso custom router to keep all routes inside the boundary. All the connections for the net must fall completely inside the fence.

The soft fence lets the Virtuoso custom router route connections both inside and outside the fence. Nets that have all their pins inside the fence are routed entirely inside the fence. Nets that have all their pins outside the fence are routed entirely outside the fence. For nets that have pins both inside and outside the fence, the fence are ignored and routes can cross the fence. A typical use for the soft fence is in a mixed-signal block to segregate the analog nets from the digital nets.

You can define fences around components as you build the cell. The routing will be localized to the components inside the hard fences.

You can have multiple fences, but they must be either hard or soft fences. The commands to work with fences are:

- *Define – Fence*

```
fence 10.5 12.25 21.5 19.0
```

- *Edit – Delete All Fences*

```
delete fence
```

- *Autoroute – Setup – Set all Fences*

```
set soft_fence off
```

The hard fence also works with the Virtuoso custom placer as a local refinement region.

Power Routing

The power router uses a different routing algorithm to complete the wires. There are many options to power routing. The ones most often used for device-level designs are the *pin_to_trunk* and *via_on_mesh* options. The power supplies can be any nets in the design.

When routing small device-level designs (0-30 devices), you may find that the power router is not useful. In this case, interactively edit the nets and define them as trunks using *Define – As Trunk*. Remember to protect the power nets after you manually route them.

To set up the power routing, select the net or wire segments and select *Define – As Trunk*, only if not previously defined.

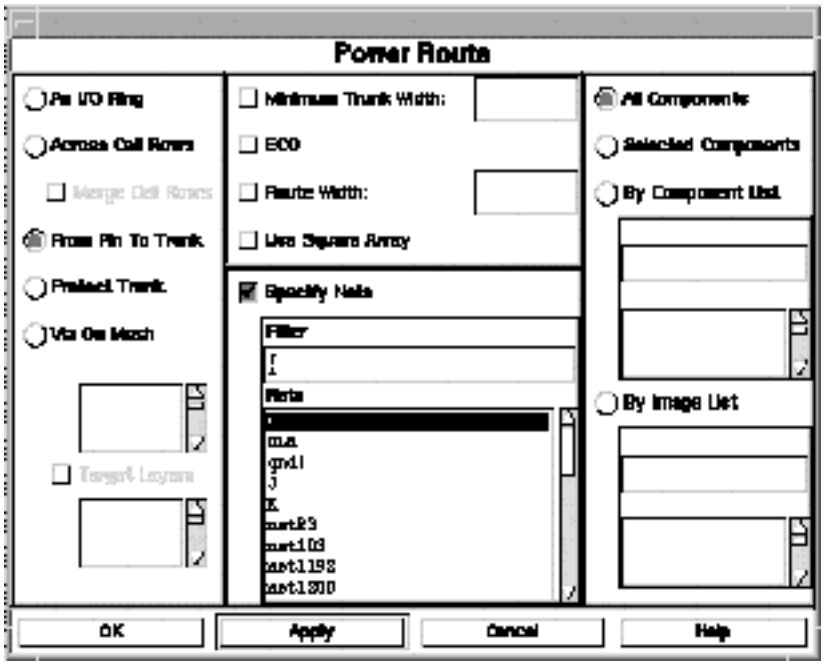
On the command line, the command example is:

```
assign_supply gndd (selected_wires)
assign_supply CLK (wide_wire (layer metal2 (min_width 5))
```

To delete the trunk information, use the menu or use the same command syntax with the `unassign_supply` command.

```
unassign_supply gndd (selected_wires)
unassign_supply CLK (wide_wire (layer metal2 (min_width 5))
```

Figure 8-3 Power Route Form



Make sure to turn off the routing layers that you do not want to use for the power supplies, such as polysilicon. You can do this on the Layers Panel or with these commands:

```
unselect layer poly
unselect layer metal3
```

The `via_on_mesh` command places a via_array where two like nets cross on different layers. You can use the `via_on_mesh` option when you have a metal2 power supply running over the top of the metal1 source pins. When you are routing power at the device-level, you may have to interactively connect the source, depending on the placement of the components.

You may want to use the *soft protect* command for the power supplies, which lets the power router move the protected wires by one wire width or space.

General Routing

General routing encompasses the remaining unrouted nets in the design. You may need to verify that the nets you have protected are not blocking any unrouted pins.

Before you start the Virtuoso custom router on the rest of the nets, save the current design. A good rule of thumb is to save a routes file (`$/$dsn_preroutes.rte`), a wire file (`$/$dsn_preroutes.w`), a placement file (`$/$dsn_preroutes.plc`), and a session file (`$/`

`$dsn_preroutes.ses`). Saving these files lets you return to a known good state very easily by starting a new Virtuoso custom router session and reading the wires and routes files. You can also load the session file as the design file in a new Virtuoso custom router session. Make sure to run the setup `.do` file that sets all the rules for the design after you reload it, if you are experimenting with the commands and do something that harms the route quality.

By following the suggestions below, you can make changes quickly and go through many iterations to achieve a very compact design:

- Always check pins for accessibility when you start a new design.
- Always run at least 25 passes of the Virtuoso custom router the first time through the design. This will ensure that the Virtuoso custom router has been given enough route passes to thoroughly attempt to route the design.
- Create wire_keepout, via_keepout, and bend_keepout areas to help guide the autorouted wires.
- Follow all route passes with a few clean passes.
- If you see any conflicts or unconnected nets, pause the Virtuoso custom router and look at the problems. You may need to interactively modify the routes then start the Virtuoso custom router again. Sometime this will be faster than just letting the Virtuoso custom router continue to run.
- After the first route and clean passes are finished, analyze the routes. If there are still conflicts, you can run the *filter* command.
- Delete wires surrounding conflicts and unconnected pins. Then make the connections with the interactive routing commands. Manually routing an unconnected net through the desired path, but not finishing the route, can help the Virtuoso custom router.
- If the route will not complete to 100%, you may have to move some components or even adjust the routing boundary.
- If you have a 100% complete route, review the poly connections to see if they can be shortened or even eliminated.
- You may need to insert diodes on the poly routes without a discharge path.
- Add custom `wiring_polygon` shapes for layers like diffusion and poly.
- Save your work periodically: write a routes file, a wire file, a placement file, and a session file. If you move any components, be sure to click the *Force Include Placement* button. This will ensure that placement information is translated back to Virtuoso XL.

Topology Editor

The Topology Editor lets you reorder the guides (flight lines), which tells the Virtuoso custom router the order in which to route the pin-to-pin connections on the nets.

The Topology Editor allows Virtual Pin insertion, which are points in the design that a wire must route. You can disqualify a pin from routing with the *fix* command. There is a From-To rules editor to define rules for pin-to-pin connections.

Routing Commands

The Virtuoso custom router has only a few commands that actually create wiring in the design. Most of the commands available are for telling the autorouter how to route the design.

Here are a few of the most common commands for device-level routing. These commands are presented in no particular order. The order in which you use them depends on your design requirements.

Automatic Data Generation Commands

These are the key commands for automatic data generation.

- **route –** This is the main autoroute command. This command takes two arguments: the total *number of passes* and the *starting pass*. This starting pass shifts the cost table to make the first pass costs equal to the *starting pass* costs. The menu command is *Autoroute – Route*. Below are some command line examples:

  ```
  route 25 1          ;# twenty five standard route passes.
  route 50 16 ;# twenty five route passes in ECO mode.
  ```

- **clean –** This command rips up and reroutes all connections except protected nets, and reroutes them with a very high cost on violations and unconnects. This command usually resolves connections the *route* command could not complete. This command is usually used with two to five passes. The *clean* command should follow a series of routing passes. When this command is being used it will remove any extra vias and wire lengths, and also improve the way it enters the pins. The menu command is *Autoroute – Clean*. Below is a command line example:

  ```
  clean 27
  ```

- **filter –** This command should follow the *route* and *clean* commands if there are still conflicts in the design. The *filter* command progressively increases the cost for conflicts for each pass. If conflicts exist after the final *filter* pass, they are ripped up and left as an unrouted or unconnected. Here is a command line example:

  ```
  filter 2
  ```

- **pin_escape –** This command routes away from the pin and insert a via at the closest possible point. You can define which layer the Virtuoso custom router will target for the via. This command works on the selected set, or all pins if none are selected. If you select a net, only the pins on the selected net are escaped. If you select a component, only the pins on that component are escaped. There is a pin select command on the *Select – Pins – Sel Pin Mode*. The menu command to route the pin escape is *Autoroute – Pre Route – Pin Escape*. Below is a command line example:

  ```
  pin_escape 2 (pin_type signal) (depth up 1)
  ```

- **Fanout –** The fanout command is similar to the pin_escape command. There is no menu entry and the command takes no options.

  ```
  fanout
  ```

- **remove_notch –** This is a post-routing command that should be run after checking finds same-net notch violations. This command makes local routing adjustments to remove same-net notch violations or adds a fill pin (wiring polygon) to remove the violation. Automatic routing functions will remove the fill pins from the design. If you have *same_net_checking* turned off, you will need to run this command after you run a *route* or *clean* command. The fill pins are saved as wiring polygons when you write a session or route file. They come back to the Virtuoso XL layout cellview as regular polygons. Below is an example of this command:

  ```
  remove_notch ;# to fill the notch
  delete all fill_pins ;# to remove the fill pins
  set same_net_checking on
  ```

Autoroute Control Commands

Several commands control routing styles and preferences independently of the rules for the routing layers and nets. These commands control the preferred direction, pin connections style, order or routing nets, and the grid. Refer to the online documentation available through the Help button for more information on the following commands.

- **set –** The set command is an environment variable for the Virtuoso custom router. Here is an example of some *set* commands for device-level routing. You can see a complete listing of the *set* values by typing `set` in the command entry area in the edit window.

  ```
  set boundary_inter_layer_check on
  set dofile_auto_repaint off
  set dynamic_zoom off
  set enforce_via_array_rule on
  set extend_wire_for_via on
  set include_pins_in_crosstalk on
  set max_gate_tie_distance 10
  set min_selection on
  ```

```
set offset_via on
set opt_clean_ratio 2
set repaint {on|off|manual}
set rotate_via on
set route_to_fanout_only off
set search_tapering on
set shrink_wire_for_via on
set soft_fence on
set trim_extension on
set write_permission {read|noread|write|nowrite}
```

Rules – Autoroute – Setup – Route

```
set force_to_terminal_point on
change escape_distance 2
```

Rules – IC – Clearance

Same Net Checking

```
set same_net_checking on
```

Taper Wire

```
rule ic ( pin_width_taper down )
```

Rules – Sorting

```
sort length up
```

Select – Vias For Routing

```
unselect all vias
select via M1_M2 M2_M3
```

Layers Panel:

```
direction metal2 vertical
unselect layer poly
```

- **max_stagger –** Sets a rule that controls the maximum segment length permitted on a layer. The rule should be only set at the layer level. Precise length control is not guaranteed; the Virtuoso custom router can add as much as twice the max_stagger length because the value is calculated from a pin, via, or t-junction.

  ```
  rule layer poly (max_stagger 4)
  ```

- **fix/unfix –** This command ensures selected nets or specified nets cannot be altered by any subsequent autorouter operations. Neither the wired nor unwired portions of a fixed net can be modified by the autorouter until you enter an *unfix* command. Wire of fixed status are treated a keepouts and cannot be involved in conflicts.

  ```
  fix net clk
  ```

```
fix class critical
fix selected
unfix selected
```

- **[un]protect –** The *protect* command prevents the autorouter from ripping-up and rerouting existing wires, IO port, well ties, diodes and vias. Use the protect command to protect preroute and other design objects that you want to preserve. Here are some command examples:

```
protect all wires
unprotect all vias
protect net CLK
unprotect net CLK
```

Here is a very simple `.do` file sample with some comments:

```
#sample do file
unit um
grid wire 0.1
grid via 0.1
bestsave on $/dsn_bestsave.w ;# save the best results
sort length up               ;# route shortest wires first
set trim_extensions on       ;# trim wire ends at pins
select net gndd vddd
tax way 50                   ;# wrong way wires expensive
route 25 1
clean 2
protect selected             ;# protect preroutes
unselect all objects
select all pins (layer poly)
fanout
unselect all objects
select all nets
unselect net gndd vddd
route 25 1
clean 2
write session $/dsn.ses
```

Device-Level Routing Tool Tips

Here are some suggestions for achieving good routing results at the device-level.

- After you have run multiple routing passes and experimented with different constraints, save a `.do` file and a session file from the current design. Quit the Virtuoso custom router

and reload the design file, executing the `.do` file for the routing and control commands. This will assure that any intermediate steps not in the current `.do` file did not skew the results.

- Time savings during routing at the device-level comes from the multiple iterations you can perform. In the time it takes to manually lay out a cell, you should be able to generate multiple *what if* scenarios in the Virtuoso custom router.
- Define rules to guide the layout process.
  - Try routing with minimal metal in the upper levels. This will help with the routability in the higher levels of the hierarchy. Use the *cost* and *tax* commands.
  - Maintain the preferred direction of the layers.
  - Set a different x and y wire grid to force porosity in the upper routing layers.
  - Placing components and pins to achieve better routes.
  - Identify the critical nets and assign rules and priorities accordingly.
- If the Virtuoso custom router fails to route a net, try to route it interactively. If you cannot, there is most likely a problem with the pin accessibility.
- Use a routing-layer-specific boundary to confine the routing. After you digitize the boundary, you can specify which layer the boundary encloses.

9

Virtuoso Compactor

Virtuoso Compactor Overview

The Virtuoso® Compactor is a symbolic compactor that compacts symbolic objects as closely as possible to achieve smallest cell size. Compaction is such that design rules are not violated in the final layout. Parameterized cells (pcells) built with ROD constructs are supported by the compactor. The compactor can handle designs up to 100k transistors using less than 900MB of memory. For more details on the Virtuoso Compactor refer to the *Virtuoso Compactor Reference Manual*.

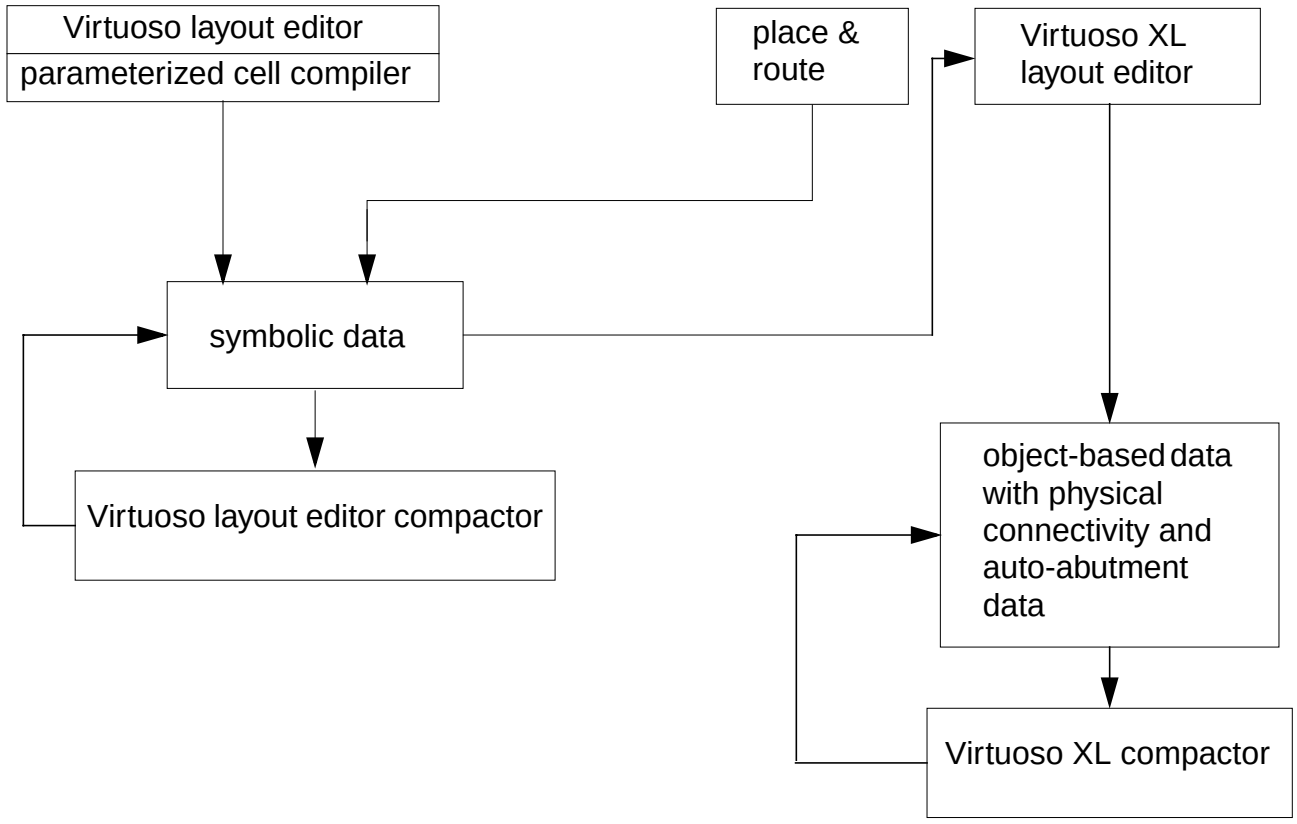
Design Creation for Compaction

Regardless of the method you use to create your layout, you can use the compactor – in either of two versions (one for Virtuoso layout editor designs and one for Virtuoso XL layout editor designs) – to compact and eliminate errors in the design.

As shown in the following diagram, the compactor can optimize a design started using any of the following Cadence tools:

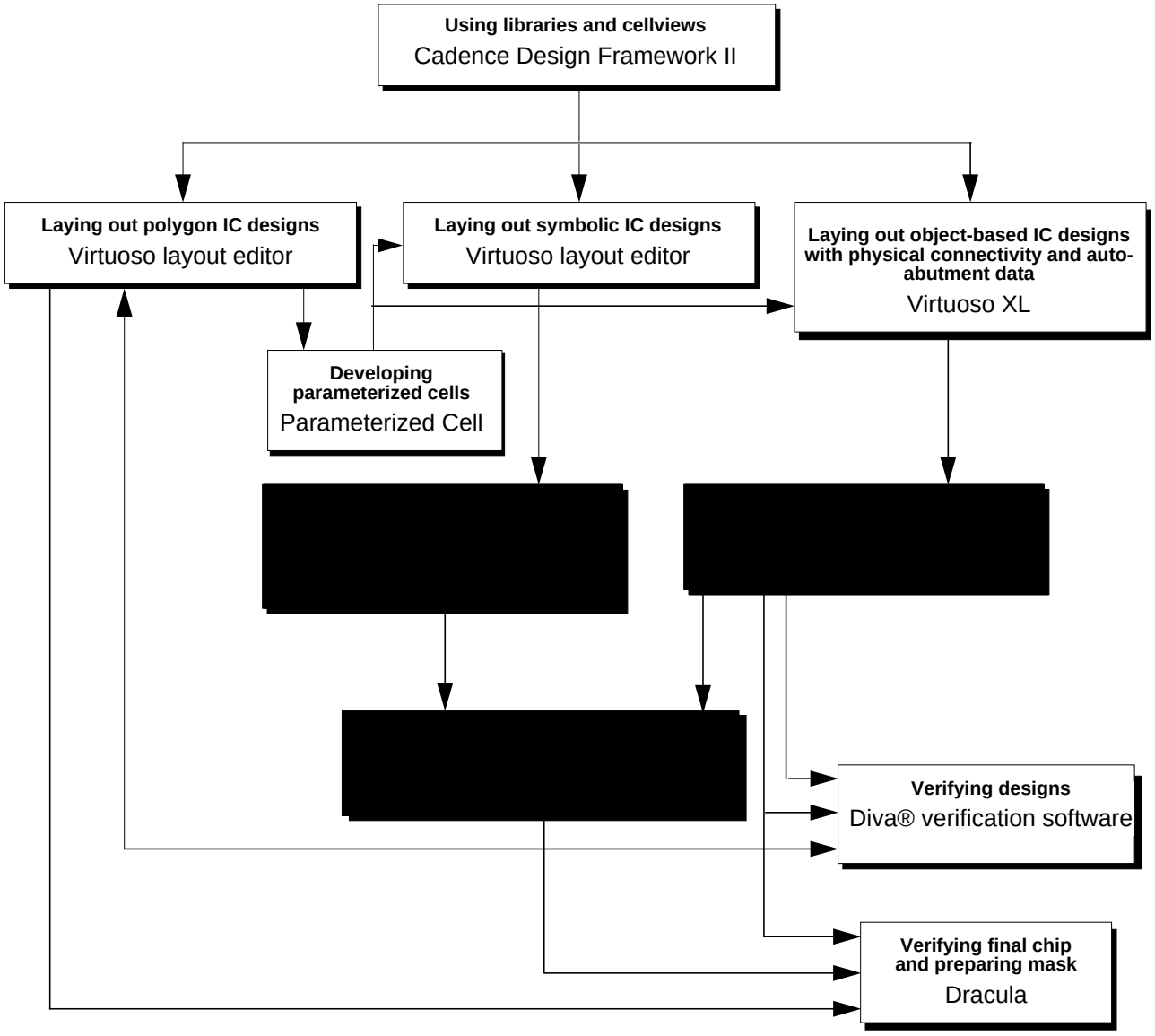
- Virtuoso layout editor, a tool for entering design data
- Virtuoso parameterized cell software, a tool for graphically defining parameterized cells
- Virtuoso XL layout editor (Virtuoso XL), a layout program that lets you generate custom layouts from schematics or edit layouts that have defined connectivity

This diagram shows which tools manipulate design data. Note the two versions of the compactor: one for compacting Virtuoso layout editor designs, and one for Virtuoso XL designs.



Note: You can feed Virtuoso XL data back into the Virtuoso layout editor and its compactor, although data processed in this way may not compact.

This chart shows the flow of the tasks involved in the layout process and the name of the product you use for each task.



Differences between Versions of the Virtuoso Compactor

While both versions of the compactor follow many of the same rules of compaction, each is tailored to support the design model of the tool it supports. The Virtuoso layout editor compactor supports abutment functions and symbolic connectivity as performed with the layout editor, while compacting Virtuoso XL designs with the Virtuoso XL compactor also

maintains and performs Virtuoso XL auto-abutment and physical connectivity. For details about Virtuoso layout editor functions, see the Virtuoso Layout Editor User Guide For details about Virtuoso XL functions, see the Virtuoso XL Layout Editor User Guide. The following table summarizes the major differences between the two versions of the compactor:

Area of Difference	Virtuoso Layout Editor Compactor	Virtuoso XL Compactor
Access	*Tools – Compactor* from the Virtuoso layout editor or Virtuoso XL design window menu banner	*Compact – Compact* from the Virtuoso XL design window menu banner
Abutment	Supports and performs abutment according to standard layout editor abutment rules	Supports and performs Virtuoso XL auto-abutment as enabled by auto-abutment parameters and Virtuoso XL option settings
Connectivity	Supports and performs symbolic connectivity according to the layout editor connectivity model	Supports and performs physical connectivity according to the Virtuoso XL connectivity model

Virtuoso Compactor Features

- Directional Design Rules

  The Compactor supports different design rules in X and Y direction, such as standard cell abutment in the X direction for channel space floorplanning and P-diffusion abutment in the Y direction to share N-wells or a power rail. You can also control the channel spaces between adjacent rows of N-diffusion. Spacing N-diffusion closer to the boundary in Y than in X allows better utilization of cell area without creating DRC violations at the block or chip level.

  Spacing rules can be specified via symRules in the technology file.

```
SymRules(
drc (boundary drawing) (ndiff drawing) (enc < 1.0) “horizontal”
drc (boundary drawing) (ndiff drawing) (enc < 0.0) “vertical”
)
```

- Manufacturing Grid Support

  The Compactor will issue warning messages if off-grid problems are caused by:

  - Design rules that are not integral multiples off grid

- User constraints that are not integral multiples off grid
- Device/pcell dimensions that are not integral multiples off grid
- Pins at off-grid positions on cells

Constraints let you control the placement of objects in the compacted result. You can set constraints before running the compactor, or if the first compaction does not yield the results you want, before recompacting.

There are three kinds of constraints:

- alignment
- separation
- magnetic

An alignment constraint forces a group of objects to align in either the X or the Y direction.

A separation constraint restricts the distance between an object and the cell boundary or between two objects. The constraint value is a pair of distances. When the lower bound of the distance equals the upper bound, the constraint is a fixed-distance constraint.

- Auto-Jogging at Wire-Pin Connections

  Auto-jogging is now possible with both area and dot pins. Auto-jogging control can be applied to layers, nets and wires. The compactor inserts jogs only to reduce area, never to reduce wire length. Enter jogs manually before running the compactor whenever they can reduce wire length or otherwise improve your design.

- Compaction of Power and Ground Taps

  Previously, vertical taps from opposite sides of a power or ground rail that align were merged and moved together. Also, auto-jogging on power and ground rails was suppressed. Now vertical taps from opposite sides of power and ground rails are allowed to move independently of each other, yielding better layout density.

- Relative Object Design Compaction

  The compactor can compact objects created using relative object design (ROD). In the case of multipart paths the compactor compacts a segment of the path. The compactor lengthens or shortens the master path and any offset subpaths, enclosure subpaths, and/or sets of subrectangles. By default, the compactor retains the length of the segments at the open ends of a ROD multipart path and compacts any segments in the middle of the multipart path. To allow the compactor to alter the length of all segments in

a multipart path, set the *syShrinkOpenEnds* property to *t*. During compaction, the compactor lengthens or shortens the ends and middle sections of this multipart path.

Using the Virtuoso Compactor with the Virtuoso XL Layout Editor

Use Virtuoso XL to edit layouts you have compacted, and then recompact the compacted layouts. Note the following when using the compactor with Virtuoso XL.

- The compactor ignores fixed constraints generated from the constraint manager. If the constraint manager is used to place a pin or a device at a specific X or Y location, the compactor does not maintain the placement of that pin or device. However, the Virtuoso compactor recognizes pins you place on the edge of the *prBoundary*. To ensure that the Virtuoso compactor keeps pins along an edge, use Virtuoso XL to fix pins along the edge of the Place and Route boundary, defined in the technology file as the prBoundary value.
- The best way to maintain logical connectivity while compacting a design generated from Virtuoso XL, is to set the *syPreservOrigNet* property on the design to *Preserve Net* before compacting.

A

Automated Custom Physical Design: Device Level Methodology Guide

This appendix is focused on presenting a documented and validated environment for accelerating device-level physical design. The documented tasks and practices discussed here represent a design environment that includes recommended solutions based on the experience Cadence® has working with customers. An important feature of this guide is that the methodology is also validated with real design data against the current Cadence product offering.

An Overview of Custom Design

Unlike regular digital designs for ASIC chips where synthesis techniques are available, the design and layout of analog/mixed signal blocks and performance critical digital blocks still require manual, labor-intensive efforts. A well defined methodology is therefore required to ensure a smooth design flow with fewer iterations and a faster turn-around.

This document presents a methodology that outlines best design practices and know-hows that are repeatable, efficient and intuitive. The methodology also encompasses the underlying tools and flows, and the technology needed to support the design process.

Following is an overview of the main phases in a custom design. A designer needs to be aware of all the phases and their inter-dependencies, irrespective of the specific tasks that he is involved with, in order to build a robust design that meets all performance requirements.

- ***Block Specification***.
  In this first step, the final block specifications are agreed upon by both the block designer and the customer. To facilitate this process, the process technology information is required.

- ***High-level Design***.
  During this step of high-level modeling and partitioning, the block is broken into sub-blocks. Constraints on these sub-blocks are derived from the agreed upon block specifications and generated.

- ***Circuit-Level Design***.
  During this step, topologies for the design at the circuit or device-level are chosen. Here, the actual transistor sizes and values for components such as resistors and capacitors are calculated. Throughout high-level modeling, partitioning and circuit design, functional verification is performed to systematically verify that any decisions being made adhere to the design requirements.

- ***Physical Design***.
  The block layout is created in this step. The layout is driven primarily by connectivity (schematic/netlist) but also needs to take in to account design constraints (area/ performance).

- ***Physical Verification.***
  The purpose of physical verification is to ensure that all electrical and design rules have been met and to make sure that the layout matches the schematic. Parasitics are also extracted for the purpose of back-annotation and simulation at this step.

- ***Postlayout Simulation.***
  In this final step, the parasitics extracted from the layout are backannotated into the design, and the complete design, including parasitics, is functionally verified.

Custom Physical Design Flow

This flow starts with schematics or schematic netlists as the primary input. However, top-level constraints such as area (die size), performance, timing and reliability requirements have to be accounted for as well. Hence, physical design should be both connectivity and constraint driven.

Floorplanning

Typical mixed analog-digital blocks contain in its analog portion bias circuits, op-amps, filters, comparators and passive components such as resistors and capacitors. The digital portion contains logic gates such as nands and nors. It is common for a digital area to use a larger portion of the design layout block. Before designing the layout of any of the sub-blocks, it is important to define the floorplan of the analog and digital portions of the block.

Floorplanning determines the areas that are more sensitive to noise and the areas allocated for interconnects. Good floorplanning is critical, and the designer should consider the following general guidelines:

- Placement of analog critical components away from the digital elements
- Short connections to the critical nodes
- Avoiding cross-over of analog biasing lines and digital buses
- Separate power for digital and analog
- Guard ring for each type of component group (analog, digital, capacitor and resistor)

Following are the pre-requisites for developing the floorplan:

- Each block level schematic must be reasonably stable (i.e., transistor sizes are not expected to change drastically)
- Block level abstracts for each of the blocks should be created based on the schematics and placed in the top-level layout
- The following factors should be well defined:
  - Placement constraints and aspect ratio for each block
  - Block I/O pin placement and power pin placement
  - Channel planning for buses (that can’t be routed over the cell)

- ❑ Power and clock planning, accounting for power drop for sensitive circuits and clock skew
- ❑ Top-level bias creation and routing
- ❑ Local and global matching considerations
- ❑ Technical risk areas for each block, e.g. signals/supplies between blocks that have noise and cross-talk sensitive nodes, parasitic sensitive nodes and circuits
- ❑ Identification of blocks to be Sea walled, and Sea wall power connections (a Sea wall is also known as a guard ring)
- ❑ Internal/ External Block metal routing conventions

When the floorplan outputs instances with fixed pin locations and boundary box sizes, designers can begin routing the top-level (or higher level) structure(s) using the block abstractions as base cells. At this level of abstraction, it is recommended that the top-level is only block based. Transistors and other device components should be grouped in the block level. The top-level schematic netlist structure should be used as a routing guide with the blocks as instance cells.

- A designer can begin to synthesize (with tool assistance) or to instantiate (manually) sub-block cells to generate all the needed instances, according to schematic instances, and generate I/O pins and estimate process boundary box.
- With all the physical components created, a designer needs to do a quick or rough component placement into the defined boundary box to verify that the provided boundary box is sufficient, and place the components consistent with the analog and digital partitioning (for noise immunity and performance) and estimate the space needed for buses, nets, and guard rings.
- The I/O pins should be placed or arranged within the boundary box. At this stage, the focus is to finalize the block geometry and place the pins in their intended locations. At the top-level of the physical design (assuming all routing metals are used), both digital and analog I/O pins must be placed around the periphery.
- To continue with other procedures (system level), the cell boundary and pin locations must be fixed. With the pin locations and boundary box size satisfied, create or save an instance for this top block cell. From the output of this step, a designer can now measure the core size and shape. The designer can then generate an abstract cell that the system design level (a higher level design) needs to begin the design. This saves time and lets designers work in parallel. At the same time, designers must commit to the fixed I/O pins and die shape and size.
- Metal Selection for Vertical and Horizontal Routing: For better routing traffic, metal layers are assigned for routing in either the vertical or horizontal direction. This is to avoid

shorting and routing traffic. In most routing practices, an odd numbered metal layer is assigned for horizontal routing, while an even numbered metal layer is assigned for vertical routing.

- Digital Top-Block Routing: Route with all the existing metal as long as it passes the design rule check (DRC). When routing digital blocks, a metal layer can be routed over the blocks. When routing over a digital block, use metal that is different from the layer that is intended to be used in the sub-block. This way, when the block is fully routed, it will not have a conflict with the top-level routing layer.
- Analog Top-Block Routing: Route with all the existing metal as long as it passes the DRC check. To avoid attracting noise to the blocks, avoid routing over the analog blocks.
- Merge Block: Merge all the abstract (or incomplete) blocks with the final placed and routed layout. Verify that the finalized block contains the same boundary box dimensions and I/O pin locations. If any of the actual physical layout does not have the same boundary box dimension and I/O pin location, then there will be a conflict between top-level block and the sub-block layout.
- A designer must either go back and fix the work previously completed in the top-level block or descend to the sub-block place and route environment to fix the sub-block. If there is any conflict between the top-level design and the sub-level design, running DRC will discover it.

Data Preparation

The purpose of this phase is to explore various architectures for the block and to begin evaluating the top-level specification and process technology imposed design constraints. Other issues that must be resolved during this phase include:

- Process limitations and feasibility of the design's critical blocks for meeting performance specifications
- Device type considerations (MOS, capacitors, resistors, inductors, parasitic bipolar transistors, diodes)
- Layout environment expectations must be considered to complete planning for them up front. Examples include additional capacitance due to fringe capacitors, general layout parasitics, electro-static discharge (ESD) considerations, electro-migration, hot-carrier effects, short-channel effects, use of guard-rings, latch up, matching problems, substrate coupling, process gradients, and etching effect.
- Specifications for interface circuitry (voltage tolerances, level shifters, input loads, output drive strengths, buffers)

Also, design method issues such as the following must be resolved:

- Identify the overall technical risk areas
- Propose and accept a simulation strategy.

The list that follows is a possible process setup for performing the physical-design task:

- Layout process technology file
- Design Library setup (where layouts of contacts and primitive cells and technology file are defined)
- DRC rule file
- Extraction rule file
- Electrical rule check (ERC) rule file
- Layout versus schematic (LVS) rule file
- Parameterized cell (pcell)
- Layer display information
- Mapping layer files (for import or export of gds format)
- High-level and circuit-level setup

The layout environment requires a layout technology file where layers, layout rules, electrical rules are defined. A library design setup, where all the layout primitive instances are defined, is also needed. For schematic-to-layout synthesis, these primitive cells are typically defined as pcells.

For layout verification, appropriate rule decks, namely a DRC rule file, an ERC rule file, an LVS rule file, and an extraction rule file, are needed. These rule files are used to verify the layout rule checks, the electrical rule checks, perform the comparison between schematic and layout and conduct the extraction of a layout netlist.

For the layout editing environment, a layer display file is required. The layout editor uses this file to display a layer's color and pattern. If this file is not provided, every layer will contain only one set of colors and patterns.

For translating one layout environment to another layout environment using the GDSII format, a mapping layer file is needed, which converts each set layer to another environment layer. For mixed simulation, the process setup in high level and circuit level is needed.

Process Migration

Often an existing design database has to be ported or migrated to a newer process technology to take advantage of the shrinking process parameters, in order to have a smaller and faster chip, without having to design from scratch. Migration can be of two types:

1. Schematic database migration (requires re-layout).
2. Layout database migration (more automated).

Device Level Layout

During this step, the goal is to transform the full schematic into a complete design layout. Because of the design-intensive nature of this step, as well as the need to consider several factors simultaneously, several iterations are often required during the process. A critical factor for iteration is the block's sensitivity to layout parasitics and other nonlinearities that are introduced during the layout phase.

To determine this, the designer must have specific knowledge of circuit design, physical design, and fabrication. The focus here is to layout all of the components and connections in the schematic as accurately as possible and to minimize parasitics on wires and components.

The first step is to create a layout template for the block that provides an overall size/aspect ratio derived from the top-level floorplan and a basic idea of where the devices should be placed, their pin locations, etc. Next, proceed with the layout of the devices in the block. During this task, the devices must be generated and floorplanned. Power, guard rings, and device placement must also be planned. Finally, the devices need to be routed. During this step, many physical and functional verification steps are performed in parallel.

Create Block Template

Prior to beginning the layout, an initial floorplan must be developed. Because of the close link between the schematic design and the physical design, the schematic often closely represents the layout topologically. This schematic, together with the floorplan developed in the previous steps, can be used as a starting point for a floorplan.

The goal of this step is to complete a first pass at placement of the major blocks so that the layout process can begin. In doing so, the designer should make an initial estimate of the area and I/O pin placement. This step need only generate sufficient information so that the block layout can be started.

Typically, estimates are based on prior experience with similar designs. Estimates can also be made by adding up the sizes of the transistors in the schematic and multiplying it by a

factor to estimate the routing area. The area of capacitor and resistor arrays can be more easily estimated based on process design characteristics.

Generate Devices

The Generate Devices step consists of the following tasks:

- Generating device components (transistors, capacitors, resistors, diodes)
- Extracting netlist information as a layout guideline
- Size estimation engine for initial block estimation
- Generating boundary box shape and I/O pins
- Building probe engine between schematic and layout
- Parameter checking at early stage (leaves fewer errors in the LVS check)

During this stage of the design, the schematics of blocks that contain device components are used as input to create the layout components. This can be done manually or automatically. Tools that do this automatically will generate the layout components, pins, netlist information and a boundary box.

I/O Pins

Before synthesizing the I/O pins, the shape, sizing, and layer of each I/O pin must be determined. The pin shape and size must be at least the minimum width and length of the layer type. At the top-level where the blocks will be instantiated, the pin layer must be the same layer as the layer type that is used to route the instance block. A conservative approach might be to use a metal 1 / metal 2 via as an I/O pin so that the layer decision can be postponed. However, care must be taken with critical nets that the additional resistance of the via does not cause problems.

Bounding Box

The bounding box definition may vary depending on how conservative or aggressive a designer wants to be. It is essential to define a good boundary box so that it provides enough room for all of the components and routing layers, without being oversized. A rule of thumb is to make the boundary box size twenty-to-thirty percent bigger then the area needed for device components, to take into account the device hook-up and routing. Another way to estimate the space for interconnection is to count the number of nets and estimate a higher percentage for higher net counts and vice versa for low net count.

Netlist

Use the netlist extraction engine or perform check against source to verify that all components in the schematic are converted to layout. Verify that all the parameters for each component match between schematic and layout. Make sure everything in schematic is converted to layout components.

Floorplan Devices

This step involves the following tasks:

- Finalize the cell box (for fixed size)
- Manually place the I/O pins in fixed locations
- Perform a quick device placement to verify and adjust the estimate box size
- Generate a fixed cell boundary box and pin location (an abstract view)

Once all the physical components have been created, the next step is to do a quick device placement into the defined boundary box to further verify that the provided boundary box is sufficient. Following the basic guidelines on whether or not it is a digital or analog component, place the devices. Also, estimate the space needed for the buses, nets, and guard rings.

Place or arrange the pin in the intended location within the boundary box. The focus of this stage is to finalize the block geometry and place the pins. For a regular digital block, I/O pins can be placed anywhere in the block; however, for analog or noise sensitive blocks, pins must be placed around the periphery.

Arranging I/O pins around the periphery affects coupling capacitance and the crosstalk that most analog blocks are sensitive to. If an adjustment needs to be made, it should be done in this stage. To continue with other steps in the flow, the cell boundary and pin location must be fixed from this point on. With the pin location and boundary box size satisfied, create an abstract cell for top-level layout design.

Plan Power, Guard Rings, and Wells

Depending on the process technology, p-well and n-well must be laid out first for areas that are intended for the NMOS and PMOS transistors. Wherever possible, join wells that can be joined together. Use separate analog and digital wells for better noise control.

Before device component placement, power rails must be planned for equally distributed power throughout the block. Power rails are most often routed over n-wells in the area where PMOS transistors are to be placed. On the other hand, the ground rails are usually routed on

the p-wells in the area where NMOS transistors are to be placed. The width of the metal used for routing power and ground must be large enough to minimize wire resistance. If the wire creates a voltage drop beyond tolerable levels, use a power strap to compensate.

A guard ring can be made of a power or ground ring. A guard ring is used to control noise from neighboring groups of components. It helps to shield the substrate from noise injected from external components. It is also the transistor bulk connectivity for forward bias protection.

After the power and wells have been laid out, the guard ring can be placed in its intended area, typically at the periphery of the block. In digital sub-blocks, guard rings are used for transistor bulk connections (for forward bias protection) and latch up protection.

Place Devices

There are two main considerations for device placement:

- Placement for better device matching (Stack placement and/or Common centroid placement)
- Placement for short RC delay interconnected route (Cells that share the same net should be placed close to one another)

In addition, spare cells should also be placed to account for Engineering Change Orders (ECOs). All placement should be DRC clean.

Placing devices or instances in appropriate locations can determine the performance and block size of the design. A designer must first design for performance before cutting corners of areas and saving power.

In analog placement, a designer must recognize the critical components. The designer must have an understanding of all the random error effects during fabrication such as undercut or edge effect, gradient or oxide thickness, and thermal gradient effects and use this understanding to carefully place the devices for device matching.

Layout of MOS Transistors

Analog MOS transistors have a large W/L aspect ratio. The layout of such elements face problems such as Oxide thickness and series resistance at source and drain. Device matching techniques such as stacked transistors and/or common centroid placement should be followed to reduce these effects.

Stacked transistor Placement:

With large W/L aspect ratio, designers sometimes choose to use non-straight or serpentine layout transistors to optimize the aspect ratio. This is not a good idea because the matching of the orthogonal elements is very poor and the contribution to the gate length of the elements at the corners is not well defined. The voltage drop due to the parasitic resistors in the chain is equivalent to an offset which cannot be neglected for precision applications. For this reason, a designer should avoid using non-straight for analog precision transistors.

To avoid problems and issues using non-straight gate transistor in analog design, a wide straight transistor is commonly split (legged) into a number of parallel and equal transistors arranged in a stack. This is to design a more manageable, better-shaped transistor, and to reduce its area. Also, a stack transistor reduces parasitic capacitance of source and drain to the substrate because of shared source and drain contacts and uses less active area.

- Instead of using one big contact over the source and drain area, use multiple contacts placed at minimum rule spacing because the risk of micro-fractures is high compared to multiple contact. Place multiple contacts at minimum spacing from the gate (poly) also reduces unwanted parasitic resistors.
- The inner gates of the stack see the same boundary (two gates at the two sides), but the gate at the end elements has a different periphery; unmatched undercut will determine slightly different lengths among the inner gate length and the outside one. For very precise designs, dummy gate is placed on both ends of the stack to make all gate boundaries result in the same undercut effect and thus improve matching.
- A common application of a stack transistor with two transistors is a differential matched pair. The two transistors are split into equal number of transistors with a common source node and are arranged inter digitally so the ratio of the two stay the same even through random errors during fabrication. Thus, breaking transistors into multiple legs and inter digitizing them in a stack layout improves the gradient effect and the averages of both transistors' characteristics are the same.
- It is good practice to layout block diagrams or block planning first before laying out a stack layout, especially a stack composed of multiple transistors. This way a designer would have a clear understanding when he begins to route. It is a good practice to draw or plan out the inter digitized blocks first before layout.
- The transistor width in stack layout should be the same. Depending on the parts into which a transistor is divided, both source and drain or the same terminal can be made available at the ending of the stack. If the different transistors have sources or drains connected to the common node, they can be combined in the stack and share contacts.
- When a designer combines a number of transistors in a stack layout, sometimes not all positions in the stack are reasonable for some transistors. The designer must then recognize which transistor or transistor pairs are most critical. The most critical, or transistors with the smallest range of dimension should be placed in the position of the stack that would most improve matching and other aspects. Transistors that have a large

range of dimension, or non-critical transistors, can be placed to meet the requirement of the stacked arrangement and achieve a well-designed layout.

- To ensure good matching, all transistors have the same orientation. The center row is where the critical transistors are planned out. The two sets of transistors are well balanced through the gradient field.

Common centroid Placement:

In a stack approach for two equal transistors, the matching strategy is to resolve a single dimension gradient effect (x-axis or y-axis). For two dimensional gradient effect (on both x and y axes), a common precise practice is to use a common centroid layout structure.

Layout of Resistors

- For resistor design, random errors are caused by thermal gradient, undercut or edge effect gradient effect due to variation in oxide thickness. These random errors can change resistor values from what a designer expects.
- Most layers in layout can be used as resistors. In analog design, resistor value can be very big or the ratio of L/W can be quite large. It can take up a lot of area in the layout. Plus the designer has to worry about the matching accuracy of the resistor due to the random errors mentioned above.
- One of the common practices for laying out a big resistor is to use the serpentine arrangement. In a serpentine arrangement, there are a lot of square corners, which have matching problems. For accurate design, round corners should be implemented into the bend area or shouldn’t be used at all.
- A straight line with low resistance wire should be used to connect all the lumps together. Straight line resistor layout is what designers prefer for precision design because each lump in a layer is the same size and is in the same orientation. All lumps should either be in the x-axis or y-axis for thermal gradient matching. Straight line resistor layout is preferred because it gives a high chance for resistor matching.
- The lumps of the resistor body at the edges do not have the undercut effect like the ones in the middle. This boundary-dependent undercut effect can lead to a mismatch in a resistor. To resolve this problem, a dummy lump of the same resistor element is placed on every edge of the resistor lump. This will get the same undercut effect throughout all the resistor lumps.
- For two equal resistor layouts in a gradient effect environment, a designer can use the inter digitized structure technique. Place each resistor lump interchangeably so the sum of the two resistors are balanced out.

- Noise can cause problems in a resistor, especially in a well resistor. Because the well layer is close to substrate, noise coupling from the substrate can effect the well resistor. A guard ring should be placed around and between the resistor lump.

Layout of Capacitor

- The causes of random errors (or mismatches) in capacitors are commonly the undercut or edge effect and gradient effect (the variation of thickness of the oxide that acts as the dielectric for the capacitor). These random effects cause changes in parameters used to calculate capacitance and should be controlled by using appropriate layout techniques.
- Layers that can generate a large capacitor value for a square unit area are layers that have the shortest distance from one layer plate to the other plate (determined by oxide thickness). For most analog layout environments, the poly2 layer is provided to be used with poly1 for generating a capacitor.
- For random errors fabrication of Oxide thickness and gradient effect, and matching of two or more equal capacitor layout, Common Centroid layout structure is commonly used. Capacitors are usually split into equal parts and placed diagonally across, so that a gradient in the Oxide thickness either in the x or y direction does not affect the capacitor matching. For a very precise capacitor design, even the wire metals are arranged in a way that the wire capacitor is equivalent between the matched capacitor terminal.
- To design a matching or proportional pair of equal capacitors, the perimeter ratio should be the same to account for the undercut effect. In addition, if the plate corner is a 45 degree corner rather than a 90 degree corner, it can improve the undercut effect. For less error, the length and width of the capacitor plate should be equal.
- To protect a capacitor from noise, a guard ring should be placed around the capacitor area. Place a dummy element next to the capacitors that are placed on the edge for improvement matching.

Noise Prevention

Noise prevention must also be considered during placement. After implications on performance specifications are thought through, the designer can consider saving area by using all available space and the interconnected delay.

- Substrate noise can be very bad for an analog block, especially if there is a lot of current flowing in the substrate. Noise is caused by substrate coupling can be controlled by using twin well process. Create a guard ring and connect the ring to the quietest reference. Much of the analog currents induced in the substrate by the digital signals will then be collected by the guard ring and will not enter the analog devices. If possible use double

guard ring, P+ and N+. Try to avoid connecting this guard ring to a reference that is used by a digital logic block.

- If single well is the only process to have to be used in designing AMS, for a sensitive circuit, try to design this sensitive circuit in the type of transistor that has a well, not directly using the substrate. A good example is a differential circuit. The differential input transistor pair is considered a sensitive signal that can be effected by the substrate noise. To avoid the substrate noise, a designer can choose the type of transistor that has well for the differential input transistor pair. In a single well, if the process is p-substrate, then PMOS transistor is more substrate noise free than NMOS, vice versa if it is n-substrate.
- A guard ring around the periphery of the analog block layout may not be sufficient enough to control noise. A common practice, analog bias signals have a series resistive decoupling and a capacitive decoupling to ground. This RC circuit helps filter out the noise that is caused by the digital module in the analog bias signals. For digital signals coming into the analog block, no additional circuit is needed.
- If a pair of power rails were to be supplied to both analog and digital modules, the noise that is generated from the digital module can cause the analog module to malfunction. Because digital modules mainly function with switches, when several switch at the same time, the power magnitude can create a bump. This bump may be tolerable among digital blocks, but it is not tolerable in analog modules. One way to avoid this power dip on VDD and ground bounce on VSS is to supply different power supplies to a digital block and an analog block (decoupling capacitor).
- Digital block activity can interfere with analog blocks if they are close to each other. If there is space available, place the analog block as far away as possible. If there is plenty of routing metal, use one of the metal to shield the entire analog block to ground. In some design cases, there is limited space and metal route. Back-End designers' best strategy is to place the high noise digital block on one corner, place the analog block on the opposite corner and the low noise digital blocks in between the high noise block and the analog block. Instead of metal shielding, use an island or a guard ring.

Layout of Digital Blocks

- In digital placement, a designer has more freedom in placing the device components. The designer need not be too concerned about device matching from the effect of random edges or gradients. The designer's main concerns are to get the smallest area placement-wise with the shortest interconnects for the least RC delays. To accomplish this, the devices can be placed so that common nets are close together, using the least amount of wire length to route to all the pins to achieve a dense and compact layout.
- In most digital designs, PMOS transistors are placed close to power rails, while NMOS transistors are placed close to ground rails. All components are typically placed to DRC minimum spacing. The more compact the placement, the higher quality the final result.

Digital blocks are usually defined with a fixed height and uniform power rails so that each gate can easily abut to another gate.

- Standard gate cell rows are usually used to place digital blocks in a row or column. Most digital layout transistor structure patterns are CMOS structures where the pattern goes vdd, PMOS, NMOS, and gnd. Because most digital standard gate cells have this common pattern, most digital cells are designed to a specific height. With each standard cell, having uniform height, power and ground rails, a designer can design a compact and dense digital block.
- For both analog and digital sub-blocks, spare cells may be needed for performance improvements during design respins, poor results from parasitic simulation, or for an ECO. Place spare cells around the devices that are critical. Adding spare cells can improve turnaround time. It can also prevent a need for redesigning the layout.

The output from this stage is the complete placement for both analog and digital blocks. The next step is to route nets to all of these device components.

Route Devices

The purpose of this step is to:

- Determine if buses are required for multiple pins
- Determine what layer is sufficient for routing
- Determine if a route is capacitance or resistance dependent
- Use the minimum metal routing layer for low level routing (digital block)
- Run DRC on sub-block routing
  - For routing the devices or instances, metal layers are recommended rather than other conductive layers such as poly. Metal layers have the smallest resistance per square area and are good conductors. Try to avoid using poly layers for routing: poly has higher resistance per square area.
  - If the signal on the net is more voltage dependent, the resistance of the wire affects the signal more than the capacitance of the wire. To reduce the resistance in the wire, make the width of the wire larger.
  - If the signal on the net is more voltage dependent, the capacitance of the wire affects the signal more than the resistance of the wire. To reduce the capacitance in the wire, make the width of the wire smaller.
  - When several pins are routed to the same net, a bus route may be required to minimize wire length so that all of the signals can be easily linked to that net.

- Digital sub-block routing usually limits the use of the existing metal layers, saving some other metal layers for top-level routing over the sub-block. Depending on the process technology, there may be three or more metal routing layers. For example, for three layer metal technology, device-level routing is constrained to a single metal routing layer, saving the other two layers for vertical and horizontal routing at the top-level.
- Because routing over analog sub-blocks is commonly not practiced due to noise, routing in an analog sub-block is not constrained to a limited number of metal layers. Use all the existing metal layers in the technology to route an analog sub-block.
- For better routing traffic, metal layers are assigned in either the vertical or horizontal direction, which is to avoid shorting and routing traffic. In most routing practices, an odd numbered metal layer is assigned for horizontal routing, while an even number metal layer is assigned for vertical routing.
- Appropriate measures should be taken for reduction of noise injection due to capacitive coupling. To prevent noise and cross talk in sensitive and critical signals, keep the lines as short as possible, decouple by using a large spacing between the victim and the aggressor and/or shield the lines.

Physical Verification

To formally verify that the layout exactly matches the schematic, the physical verification process checks the layout against the design and electrical rules. The physical verification tasks are described below in the order they must be performed.

While the first verification step, DRC, is an independent task, the subsequent steps such as ERC and LVS require extracted netlist information; therefore, once the layout is DRC clean, a netlist must be extracted from the layout. This netlist information is then used for ERC, LVS and parasitic simulation.

The ERC and LVS extraction netlist does not usually need to contain parasitic information. But for parasitic simulation, a netlist must be extracted with parasitic resistance and capacitance information for all layer characteristics. While this step is not a verification step in and of itself, it is necessary for later verification, and it does check for netlisting syntax.

The Design Rule Check (DRC)

The DRC is the first verification task and should be run either during layout or after layout is complete. DRC checks the layout to see if layers are being placed according to the rules specified by the process manufacturer. Although there is no real dependency between DRC

and extraction, it is important to fix all DRC errors prior to proceeding with each of the verification steps.

Design Rule Check (DRC) Basics

The Design Rule Check is a part of layout verification. The DRC checks layout designs to see if they satisfy the constraints imposed by the fabrication process for a selected technology.

No matter how conscious a designer is of the design rules when performing the layout, they are often overlooked. In general, before designs can be manufactured, they must pass DRC. The quality of the DRC check depends on the DRC rule file.

A quality DRC rule file contains several rule checks such as antennas, width, spacing, dog bone, overlap, esd, fat wire, notch, grid, digital and analog, etc. Different fabrications have different DRC rules, even if the technology is the same.

Sometimes, when a designer routes a design using a layout tool, the wire does not appear to be shorted to another wire or opened, but after the design is fabricated, the layers are sometimes shorted to another layer or opened. Passing a DRC check guarantees the following:

- The layout will not be shorted or opened because drawn layers are too close or the layer is too thin
- Each transistor type is assembled correctly from a set of layers
- What is output from the layout tool will be masked and etched accordingly

During placement or routing, run as many DRC checks as possible. Don’t wait for the entire placement or routing to be completed, because one error can create a domino effect for the whole design, especially if the error is at the center of the design.

A designer may need to readjust other cells or routed wires to correct that particular cell.The sections that follow outline DRC checks that must be performed during various stages of the design.

1. DRC on Block Plan: Run a DRC check on the completed block layout. At this particular step, the DRC verification check is to verify that the layout of the power plan, guard ring plan, and well plan are DRC clean before transistor placement proceeds.
2. DRC on Block Placement: This DRC verification check is to check transistor layout cells and ensure that all the layers that make up of transistor (poly, diffusion, metal1, via and cut) will not violate DRC rules. This step also ensures that cell spacing is DRC clean and verifies that these transistor cells are DRC clean when placed with the guard ring and power grid. It is important to remember that running DRC after completing each step

guarantees catching physical errors as soon as possible, rather than waiting until the placement and routing step has been completed and thereby eliminating concerns about layout work failing DRC at any step.

3. DRC on Route: This step is to ensure the DRC correctness of the routing.
4. DRC on Merge: Running a DRC check at this step provides verification of the overall physical design, particularly in the layout interface between the top-level and the lower block level. This is the layout close to the boundary box of instance blocks from both the top-level and the block level. This area is where DRC has been checked. Running DRC at this step is the final DRC check. Verify that every physical layout is DRC clean. All DRC errors must be resolved before proceeding on to the next step. A DRC error can lead to a misconnected, short or open circuit.

The Electrical Rule Check (ERC)

When the layout work is complete and passes DRC, the electrical rule check (ERC) is run to check for open and short circuitry. The ERC check is primarily for global checking, such as a power connection check, and is used for the following:

- To check the physical layout electrical syntax
- To perform a fast circuit analysis

The Electrical Rule Checker (ERC) is a quick and time saving analysis that checks the electrical rules of the layout circuit. ERC is similar to a simulator checker, checking the circuitry netlist for errors such as undriven nodes, unused transistors, floating nodes, shorting nodes, and power/ground nodes.

ERC should be run prior to LVS, because if the physical design does not pass the ERC check then it is of little value to run LVS. If there are errors in the ERC check, then there are bound to be errors in the LVS check. It is best to pass ERC prior to running an LVS check, leaving other mismatches that ERC is not able to detect for the LVS check. ERC errors, if not fixed while running LVS, will give some false errors.

Because of this, it is best to run ERC prior to LVS, especially when dealing with large-scale designs. LVS can check what ERC can, but it would be difficult to troubleshoot the errors, especially with false errors.

The quality of the ERC check depends on the ERC rules. To ensure a successful ERC check, verify that your rules are comprehensive and complete. An ERC check should be run at both the sub-block and block levels, to reduce complexity and to fix any problems up front.

Layout Versus Schematic (LVS)

At the Layout versus Schematic (LVS) stage, the layout circuitry is compared to the schematic circuitry. This is one of the main verification steps that determines if the layout has the same circuitry as the schematic. The LVS step provides the designer with feedback as to whether or not the layout matches the schematic.

LVS requires an extracted netlist and compares the following areas:

- Connectivity of layout and schematic
- Parameter values between schematic and layout
- Number of components between schematic and layout

The LVS check is a logical verification between two netlists. One netlist is extracted from the schematic and the other is extracted from the layout. LVS compares the two netlists to see if they are the same. LVS can compare at the instance or transistor level. It compares nodes and parameter sizes.

Usually, during LVS verification, schematic information is used as a reference. Any errors that occur during an LVS check should be fixed in the layout to match the schematic.

The quality of LVS verification depends on the LVS rules. Good LVS rules would give quality LVS verification results. An LVS check should be run at both the sub-block and block levels, to reduce complexity and to fix any problems up front.

Troubleshoot Layout Versus Schematic (LVS) Errors

The first approach in troubleshooting the LVS is to verify that all of the basic components between schematic and layout match (not counting parallel components mismatched). If component matches are not resolved, then other errors such as in the net, terminal and parameter may be the cause of this mismatch.

After components are matched, the second approach is to make sure the connections and terminals match between layout and schematic. This mismatch or match does not affect the parameter comparison, but it is more important. If connections are mismatched and the functionality is mismatched between schematic and layout, then the design has already failed.

The next important area to troubleshoot would be to verify that the parameters between the schematic and layout are matched. When the parameters match, the layout is then considered a matched design.

Netlist Extraction for Layout Versus Schematic and Electrical Rule Check

The purpose of this step is to extract the netlist layout without parasitic for LVS and ERC verification. Use the extractor to extract only devices (and their parameters) and connectivity from the layout. The netlist is used either to compare the schematic versus layout (LVS) or electrical rule check (ERC). For this reason, parasitic information should not be included in the extraction procedures.

Note: Do not use a parasitic extraction netlist to run LVS with a schematic netlist. Because the two circuits are different, this would cause errors. If the schematic and layout have similar parasitic structures, then it would make sense to run LVS with parasitic netlist.

Parasitic Netlist Extraction

The purpose of this task is:

- To extract device components with resistance and capacitance parasitics
- To prepare the layout netlist characteristics for postlayout simulation

At this stage of the design, the LVS comparison between schematic and layout must be clean; or else, when running postlayout simulation with this stage, the output netlist may give unexpected results. The extraction in this stage is to generate a netlist with true layout characteristics. This means device components, connections, and parasitic components must be extracted.

The device components and connections are the intentional components that are already modeled in the schematic. The unintentional components that may exist in the layout design can be resistance, capacitance, diode, and BJT. These unintentional components can be considered “noise components” because they will generate unwanted characteristics. Postlayout simulation is needed because parasitic components can change the design performance.

The accuracy of the data depends on the extraction rules. The designer may need to check design rules and make sure that the equations in the rules are correctly written. Inaccurate equations may lead to inaccurate component values. Also verify that each layer the rule uses is the intended one. A parasitic net list extraction should be run at both the sub-block and block levels, to reduce complexity and to fix any problems up front.

Exclusive OR (Optional)

Exclusive OR verification is usually performed when a design is 95 percent complete, or when a respin design needs only a few fixes of layout work. Exclusive OR is used to verify that the

modified area of design is highlighted when compared to the layout data before modification. This ensures that the designer did not accidentally place a layer in an unintended area.

Postlayout Parasitic Simulation

Parasitic simulation is the final step of layout verification and takes place prior to sending the design to manufacturing. This simulation is required because the netlist extracted from the layout design has parasitic or noise information data because the circuit is now more fully implemented. This netlist simulation provides the designer with the actual (when the design is fabricated on silicon) circuit characteristic. Hence, once layout is complete and the layout verification is clean, extract the layout circuit including the parasitic of the layout to a transistor netlist file and run netlist simulation. The purpose of post-layout simulation is to:

- Compare layout design performance to schematic design performance in order to verify that the functionality of the layout matches the schematic.
- Determine layout design performance by simulating with the parasitics of wire layers and transistor gate, drain and source parasitic capacitances etc. that were not in the schematic stage
- Determine if the layout design meets the custom design specification
- Determine circuit adjustment needs

To maintain consistency between the schematic and layout design phases as well as facilitate pinpointing the cause of any problem, use the same verification setup, testbench, and simulation strategies that were used during the schematic design phase to simulate the layout design. Also, to obtain similar output results, the schematic phase input control strategies must also be used during the postlayout simulation.

To determine whether or not the postlayout simulation data meets the specification, a designer should collect the postlayout simulation data and compare it to the block design specifications. If the simulation data meets the design specification, then the design has met the goal. If the data does not meet the design specifications, then the designer must return to the routing procedure to see if the parasitic component value can be reduced to reduce the noise.

If the routing procedure cannot be improved upon, then the designer should tweak the placement procedure. If that doesn't work, then go to the circuit design stage and modify or add more strength, or extra functions, to improve the performance. The pattern for improving the performance of the design, when it fails to meet the goal specification, is to go to the previous design stage, and so on, until the problem is resolved.

B

Automated Custom Physical Design Example

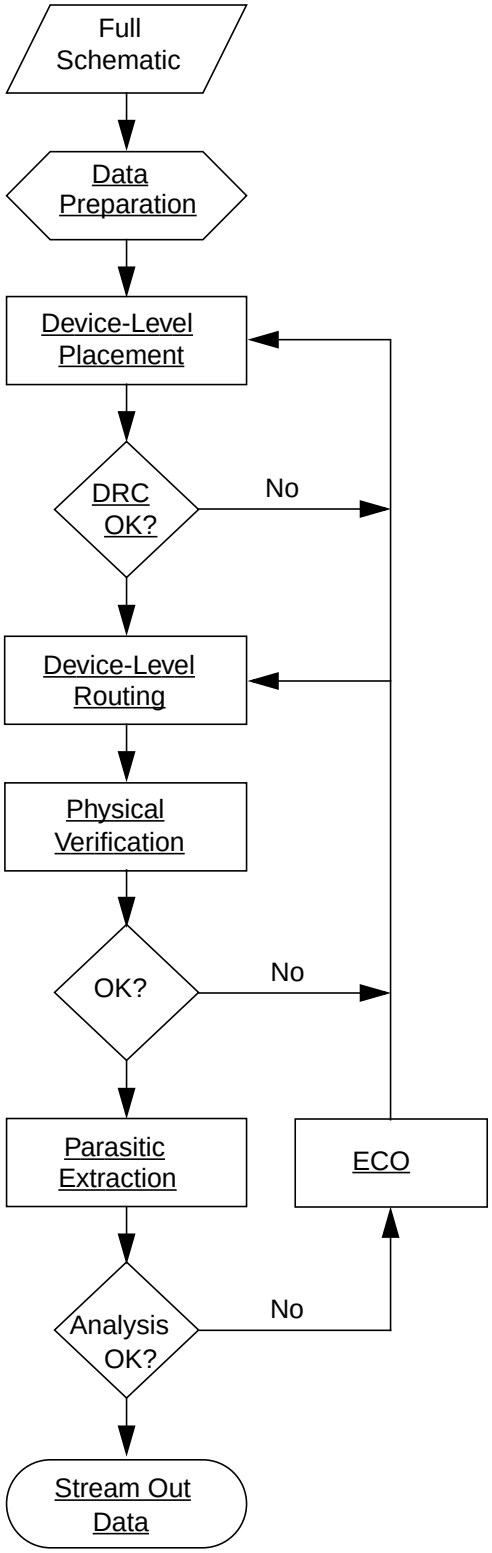
Methodology Overview

This appendix describes, in general terms, the full design cycle for a device-level physical design that meets the needs of a custom or semicustom layout. Some engineering-related task are not covered in detail but are expected to be consistent with general industry practice.

Unlike regular digital designs for ASIC chips, where synthesis techniques are available, the design and layout of analog/mixed-signal blocks and performance-critical digital blocks still require manual, labor-intensive efforts. A well defined methodology is therefore required to ensure a smooth design flow with fewer iterations and a faster turnaround.

Following is an overview of the main phases in a custom device-level physical design. You need to be aware of all the phases and their interdependencies, irrespective of the specific tasks that are involved.

The flow diagram illustrates the steps to create a complete custom IC physical design solution (click on a link to view more detailed information):


Full Schematic
Data Preparation
Device-Level Placement
DRC OK?
No
Device-Level Routing
Physical Verification
OK?
No
Parasitic Extraction
ECO
Analysis OK?
No
Stream Out Data

Engineering Input

The physical design flow starts with Virtuoso® Schematic Composer schematics or schematic netlists as the primary input. Topologies for the design at the circuit or device-level are chosen during the circuit design phase. The actual transistor sizes and values for components such as resistors and capacitors are calculated. Throughout high-level modeling, partitioning, and circuit design, functional verification is performed to systematically verify that any decisions being made adhere to the design requirements. This information will be used to estimate the physical size of the design.

The schematic helps determine the placement of the layout. Top-level constraints – such as area, performance, timing, and reliability requirements – have to be accounted for as well. Physical design should be both connectivity and constraint driven.

The ROD Sample Parameterized Cell Library

The relative object design (ROD) sample parameterized cell (pcell) library contains layout views of devices built using ROD functions. A pcell is a graphic, programmable cell that lets you create a customized instance each time you place it. The pcells in this collection serve as examples of how you can use ROD functions to create your own pcells.

Exploring the Design

This phase explores various architectures for the block and begins evaluating the top-level specification and process technology-imposed design constraints. Other issues that must be resolved during this phase include

- Process limitations and feasibility of the design's critical blocks for meeting performance specifications
- Device type considerations (MOS transistors, capacitors, resistors, inductors, bipolar transistors, diodes)
- Layout environment expectations must be considered to complete planning for them up front; examples include additional capacitance due to fringe capacitors, general layout parasitics, electrostatic discharge (ESD) considerations, electromigration, hot-carrier effects, short-channel effects, use of guard-rings, latch-up, matching problems, substrate coupling, process gradients, etching effect, etc.
- Specifications for interface circuitry (voltage tolerances, level shifters, input loads, output drive strengths, buffers, etc.)

Also, design method issues such as the following must be resolved:

- Identify the overall technical risk areas
- Propose and accept a simulation strategy

This discussion is completed in much more detail in "Data Preparation" on page 122.

Device-Level Physical Design

Device-level physical design transforms the full schematic into a complete design layout. Because of the design-intensive nature of this step and the need to consider many factors simultaneously, several iterations are often required during the process. A critical factor for iteration is the block's sensitivity to layout parasitics and other non-linearities that are introduced during the layout. The focus here is to lay out all the components and connections in the schematic as accurately as possible and to minimize parasitics on wires and components.

The first task is to create a layout template for the block that provides an overall size/aspect ratio derived from the top-level floorplan and a basic idea of where the devices should be placed, their pin locations, etc. Next, proceed with the layout of the devices in the block. During this task, the devices must be generated and floorplanned. Power, guard rings, and device placement must also be planned. Finally, the devices need to be routed. Physical verification should be performed often at various stages of device layout to ensure DRC correctness and to reduce the number of iterations by catching errors early in the layout process.

Device-Level Floorplanning and Placement

You can begin by synthesizing (with tool assistance) or by instantiating (manually) devices/cells to generate all the needed instances, according to schematic instances. You will also need to generate the I/O pins and estimate the process boundary box.

With all the physical components created, you need to do a quick or rough component placement into the defined boundary box to verify that the provided boundary box is sufficient. This placement of the components must be consistent with the analog and digital partitioning (for noise immunity and performance) and should allow for the space needed for buses, nets, and guard rings.

The I/O pins should be placed or arranged within the boundary box. The focus is to finalize the block geometry and place the pins in their intended locations. At the top-level of the physical design (assuming all routing metals are used), both digital and analog I/O pins must be placed around the periphery.

To properly align with other blocks in the design (system level), the cell boundary and pin locations must be fixed. With the pin locations and boundary box size satisfied, create or save an instance for this block. You can now measure the core size and shape. Use the measurement to generate an abstract cell that the system design level (a higher-level design) needs to begin the design. You must commit to the fixed I/O pins and die shape and size.

This is explored in more depth in "Device-Level Floorplan and Placement" on page 125.

Running the design rule check (DRC) checks to ensure that placement meets with the design criteria is a recommended step after the proposed placement is completed.

Device-Level Routing

Once placement has been completed, the next step is to connect all the devices and components. A good footprint and placement is key to achieving optimal routing results.

The routing layers to be used to route the block should be determined before routing. For better routing traffic, metal layers are assigned for routing in either the vertical or horizontal direction. This helps to avoid shorts and to manage routing traffic. In most routing practices, an odd numbered metal layer is assigned for horizontal routing, while an even numbered metal layer is assigned for vertical routing.

In digital blocks, higher layer metals are typically reserved for top-level routing and are not available at the device-level. For analog/mixed-signal blocks, all available metal layers might be used because routing over the top of these blocks is generally avoided. If required, tracks for feedthroughs should be reserved.

It is crucial to decide on a power routing strategy for the block. Often an existing power template (as dictated by top-down footprint) is used for the purpose. Critical nets also need to be identified with non-minimum width, special spacing, shielding, and other such requirements in mind. These and any other electrical constraints that are defined in the top high-level specification need to be implemented.

Once the routing has been completed, the results should be analyzed and iterations made. DRC checks should also be performed on the routed block.

This is explored in more depth in "Device-Level Routing" on page 140.

Physical Verification

Physical verification ensures that the design layout meets the process requirements for the purpose of fabrication and implements the circuit correctly. It generally consists of the following checks:

- Design rule check (DRC)
- Netlist extraction (for ERC/LVS)
- Electrical rule check (ERC) [Optional]
- Layout versus schematic (LVS)
- Exclusive OR (Optional)

More information on this topic is available in "Physical Verification" on page 154.

Parasitic Extraction and Backannotation

The purpose of this task is to

- Extract device components with resistance and capacitance parasitics in order to prepare the layout netlist for postlayout simulation
- Backannotate the schematic with parasitic information for the purpose of simulation or documentation

More information on this topic is available in "Parasitic Extraction" on page 173.

Postlayout Analysis

The outputs from the layout step will be the extracted layout and the backannotated schematics, which can be used for postlayout simulation. These will be used to verify performance and to ensure reliability.

The analysis is based on the parasitic data extracted from the design. It can uncover design problems and may require changes in the layout. These potential changes are covered in "Design Modification" on page 180.

Engineering Change Order

Engineering design changes needed as a result of the analysis, are documented in an ECO. The physical layout tools have special features that are designed to aid in preserving the completed and unaffected work while making these changes. You will have to study the changes to determine the impact that they might have on the design. Minimizing the changes also minimizes the chances of having further problems with postlayout analysis. These potential changes are covered in "Design Modification" on page 180.

From time to time, ECOs will also be used to make design improvements or other changes even after the product has been fully completed and shipped to customers. These same techniques can be used to minimize the impact of these changes.

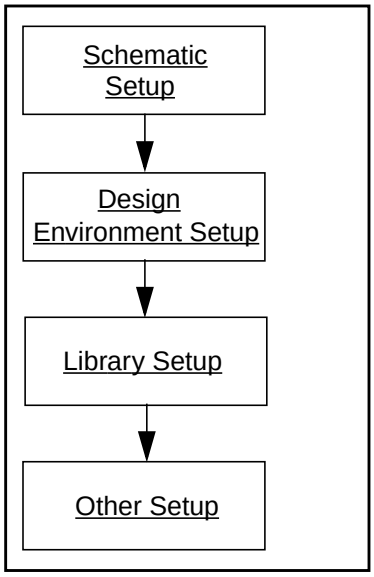
Final Steps

Once the layout of the design is fully complete, a General Data System (GDS) output needs to be generated that can be used to fabricate the chip (and/or for export to another environment). Consult “Stream Out the Data” on page 176 for more specific details on this step. Once the design is translated and a test of the fabrication process has been completed, the design can be archived.

Data Preparation

Data preparation requires an understanding of the engineering design and the environment.

The following flow diagram illustrates the steps for data preparation (click on a link to view more detailed information):



Study the Engineering Design

The first phase of the physical design and verification process is to evaluate and design constraints imposed by the top-level specification and the process technology. Issues that would typically be resolved during this phase include

- Process limitations and feasibility that the design's critical blocks will meet performance specifications
- Device type considerations (MOS transistors, capacitors, resistors, inductors, bipolar transistors, diodes)
- Electrical issues such as noise, cross-talk, IR drop, and power and ground bounce
- Physical effects such as parasitics, electrostatic discharge, electromigration and self heating, hot-carriers, alpha particles, short-channel effects, latch-up, matching problems, substrate coupling, process gradients etc.
- View the topology as represented by the Engineering schematic and understand the critical requirements

Once the issues, constraints, and risks have been identified, the physical design process can start with the schematics as the connectivity source. For a detailed discussion of the schematic considerations as they effect the layout, refer to "Schematic Setup Recommendations" on page 186. The schematic is the key communications vehicle from the design engineering to the layout designer.

Setup Requirements for Physical Design

The list that follows is a possible process setup for performing the physical design task:

- Environment setup/initialization files
- Layer display information
- Process technology library/file
- Design library
- Translation rule files for automatic placer and router
- Rule decks for DRC, ERC, LVS, and parasitic extraction
- Mapping layer files (for import or export of GDS format)
- Simulation setup

The layout environment requires a technology file where layers, physical rules, electrical rules, etc., are defined. A design or technology library setup, where all the layout primitives are defined, is also needed. For schematic-to-layout synthesis, these primitive cells are typically defined as pcells (parameterized cells). In addition, certain environment setup files are also required.

For layout verification, appropriate rule decks – namely a DRC rule file, an ERC rule file (optional), an LVS rule file, and an extraction rule file – are needed. These rule files are used to verify that the layout meets the rule requirements for DRC, the electrical rules, and to perform the comparison between schematic and layout. Rule decks are also used to guide the extraction of a layout netlist.

For the layout editing environment, a layer display file is required. The layout editor uses this file to display a layer's color and pattern. If this file is not provided, every layer will contain only one set of colors and patterns.

For translating one layout environment to another layout environment using the GDSII format, a mapping layer file, which converts each set layer to another environment layer, is needed.

Refer to "Data Preparation Detailed Instructions" on page 186 for detailed information.

Process Migration/Rereferencing

Often an existing design database has to be ported or migrated to a newer process technology to take advantage of the shrinking process parameters, in order to have a smaller and faster chip, without having to design from scratch. Migration can be of two types:

- Schematic database migration (requires re-layout but allows IP reuse)
- Layout database migration

Device-Level Floorplan and Placement

The goal of this step is to create an initial floorplan that is then used to create a layout footprint, generate all the layout components, and place them optimally to minimize interconnect lengths. Any top-down requirements such as I/O pin order and positions as well as a power plan will need to be taken into consideration. High-level constraints such as device matching, symmetry, and noise have to be factored in to the placement, as well as ensuring short net lengths for critical signals. The area utilization should be low enough for routing purposes without being too pessimistic. Spare tracks and feedthroughs should be accounted for, if applicable.

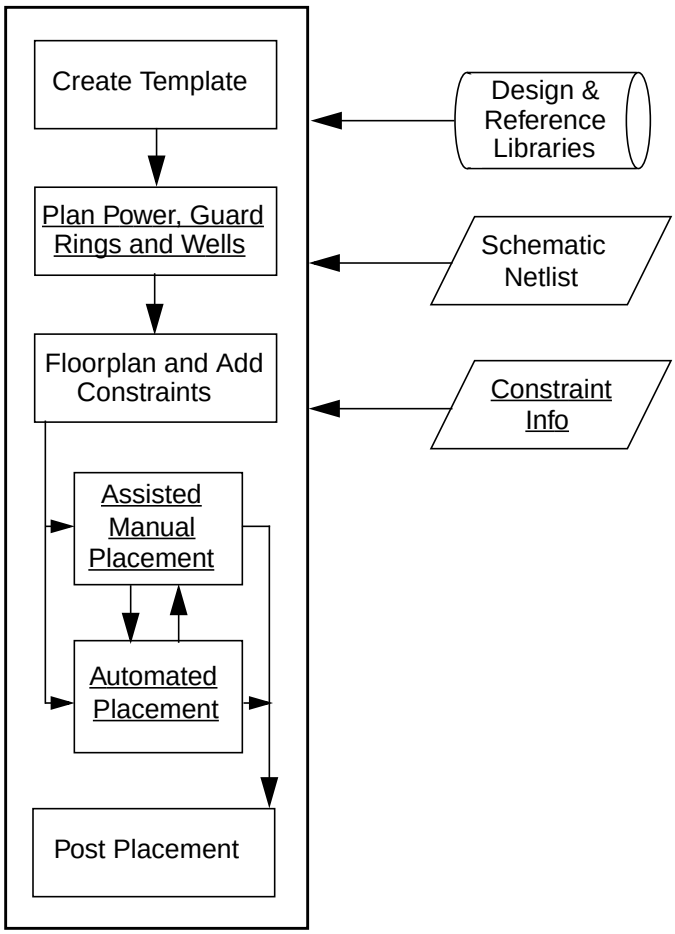
The connectivity-driven layout design tool Virtuoso XL Layout Editor (Virtuoso XL) is used for generating and placing layout components. Virtuoso XL offers a wide range of layout design functionalities, including

- Automatic generation of pins and instances
- Automatic sharing of diffusion (autoabutment)
- Automatic folding and chaining of MOS devices

Being a connectivity driven layout tool, Virtuoso XL retains schematic connectivity. It can perform checks between the schematic and the layout to make sure they are always synchronized, and updates the connectivity when there is a change in the source schematic (in an ECO situation). Virtuoso XL is highly productive and efficient, providing significant automation and built-in intelligence to work successfully in a manual custom physical design environment.

Components can also be floorplanned and placed in an automated manner using the Virtuoso Custom Placer. The Virtuoso custom placer is the device/standard cell placement tool tightly integrated in the Virtuoso XL environment. The Virtuoso custom placer generates placements very quickly and can be used to perform ‘what-if’ floorplanning/placement analysis and as the starting point for production quality placement. This represents a major productivity enhancement for floorplanning/placement.

The following flow diagram illustrates the steps for floorplanning and placement (click on a link to view more detailed information):



Create the Footprint

Prior to beginning the layout, a footprint should be developed. Because of the close link between the schematic design and the physical design, the schematic often closely represents the layout topologically. This schematic, together with the top-level floorplan, can be used as a starting point for a footprint. The goal of this step is to generate a footprint and do a first pass placement of the devices, so that the layout process can start.

In order to generate the footprint, a designer should make an initial estimate of the area and I/O pin placement. Typically, estimates are based on prior experience with similar designs. Estimates can also be made by adding up the sizes of the transistors in the schematic and multiplying it by a factor to estimate the routing area. The area of capacitor and resistor arrays can be more easily estimated based on process design characteristics. Virtuoso XL provides

the user options for creating a footprint in the Layout Generation Options form for layout generation.

Creating a footprint consists of the following tasks:

- Generating boundary box shape and I/O pins
- Generating device components (transistors, capacitors, resistors, diodes)
- Extracting netlist information to be used as a layout guideline
- Size estimation for initial block estimation
- Parameter checking at early stage (leaves fewer errors in the LVS check)

The above steps are typically performed in Virtuoso XL using the *Gen from Source* command. The parameter comparison between schematic and layout can be done with the help of *Connectivity – Check – Against Source* command.

Bounding Box

The bounding box definition may vary depending on how conservative or aggressive a designer needs to be. It is essential to define a good boundary box so that it provides enough room for all of the components and routing layers, without being oversized. Make the boundary box size 20 to 30 percent larger than the area needed for device components to allow for the device hookup and routing. Another way to estimate the space for interconnection is to count the number of nets and estimate a higher percentage for higher net counts and vice versa for low net count. Be sure and anticipate the need for shielding (guard rings) and noise suppression in the estimate.

By default, Virtuoso XL computes the total area of the bounding boxes of the individual layout components and calculates the boundary box accordingly, based on the aspect ratio or utilization factor provided by the user in the Layout Generation Options form.

I/O Pins

Before synthesizing the I/O pins, the shape, sizing, and layer of each I/O pin must be determined. The pin shape and size must be at least the minimum width and length of the layer type. This is ensured by the default settings for pin sizes in the Layout Generation Options form. At the top-level (where the blocks will be instantiated) the pin layer must be the same layer as the layer type that is used to route the instance block.

Place or arrange the pins in the intended locations within the boundary box based on top-level floorplan constraints. Typically the pins are placed on the periphery of the block but

sometimes that may not be possible due to congestion. For a regular digital block, I/O pins can be placed anywhere in the block; however, for analog or noise-sensitive blocks, pins must be placed around the periphery. Arranging I/O pins around the periphery will minimize capacitance and the crosstalk which are typically problems within analog blocks.

At the top-level the I/O pin placement order and in some cases specific locations may be dictated by the specification. Generally this is required for concurrent design of blocks in a top-down design environment. In a purely bottom-up design situation, the pin placement is a decision that can be made during the layout process.

The Layout Generation Options form allows pins to be generated automatically from the schematic in user-defined layers and sizes (including additional pins for spares and feedthroughs), while the Virtuoso custom placer provides automatic pin placement capability.

Generating Device Components

During this stage of the design, the schematics of blocks that contain device components are used as input to create the layout components. A layout device corresponding to every schematic device needs to be generated in order to verify the estimated block size and to understand the connectivity and other constraints. Components generated could be active devices, such as MOS transistors, or passive elements, such as resistors and capacitors. Once again, the *Gen from Source* command allows users to generate device instances automatically with options such as folding and chaining (for legging wide MOS devices and to share diffusion when applicable).

Extracting Netlist Information

Use the netlist extraction engine or perform check against source to verify that all components in the schematic are converted to layout. Also verify that all the parameters for each component match between schematic and layout by choosing the *Connectivity – Check – Against Source* command.

See “Create Footprint Detailed Instructions” on page 190 for specific information on these steps.

Plan Power, Guard Rings, and Wells

Before final device/component placement, power rails must be planned to permit equally distributed power throughout the block. Power rails are most often routed over n-wells in the area where PMOS transistors are to be placed. In a similar manner, the ground rails are usually routed on the p-wells in the area where NMOS transistors are to be placed.

The width of the metal used for routing power and ground must be large enough to minimize wire resistance. If the wire creates a voltage drop beyond tolerable levels, use a power strap to compensate. (One way to avoid this power dip on vdd and ground bounce on vss in a mixed- signal block due to simultaneous switching of digital gates is to supply different power supplies to digital and analog modules. Decoupling capacitors should also be added wherever possible to mitigate power dip and ground bounce.)

Depending on the process technology, p-wells and/or n-wells should be laid out first for areas that are intended for the NMOS and PMOS transistors, respectively. Wherever possible, join wells that can be joined together. Use separate analog and digital wells for better noise control.

It is often necessary to use an existing power footprint. This can be done in Virtuoso XL by instantiating a (top-down) footprint and flattening it while retaining the original connectivity information. Power rails can be drawn in Virtuoso XL or in the Virtuoso Custom Router as a polygon. It is also possible to define the supply rails within the automatic placer, if you are using it for row-based automatic placement.

A guard ring can be made of power or ground. A guard ring is used to control noise from neighboring groups of components and to shield the substrate from noise injected from external components. It is also the transistor bulk connectivity for forward bias protection. After the power rails and wells have been laid out, the guard ring can be placed in its intended area, typically at the periphery of the block. In digital sub-blocks, guard rings are used for transistor bulk connections (for forward bias protection) and latch-up protection. Guard rings can easily be drawn in Virtuoso XL using the ROD-based multipart-path option.

Note: A guard ring around the periphery of the analog block layout may not be sufficient to control noise. As a common practice, analog bias signals have a series resistive decoupling and a capacitive decoupling to ground. This RC circuit helps filter out the noise that is caused by the digital module in the analog bias signals.

See “Power, Guard Rings, and Wells Detailed Instructions” on page 195 for detailed information on these steps.

Device Placement

The inputs to this step are

- A cell or a block footprint
- A power plan including guard rings/wells

You will need to ascertain the best placement of the devices based on connectivity and performance constraints. All placements should be DRC correct. The block geometry may be

updated based on this actual placement. To continue with other steps in the flow, the cell boundary and pin location must be fixed from this point on and an abstract cell should be created for top-level layout design. Because devices are not yet routed or hooked up, there may be more iterations to attain a better placement-based on the routing results.

Certain Virtuoso XL commands can be used effectively for device-level placement. These commands include cross-probing between schematic and layout, viewing the connectivity with the help of flight lines, aligning groups or clusters, and being able to instantiate existing and flatten them in the layout while preserving the connectivity information.

The Virtuoso custom placer can also be used very effectively to do a ‘what-if’ analysis of placement. This analysis includes computing the number of rows for a given maximum transistor width/cell height, horizontal and vertical utilization of the rows, and overall utilization for a given row height and spacing. Analysis of trial placement results can help arrive at an optimal block size and aspect ratio as well as optimal folding thresholds for the transistors. You can save different iterations of the placer information to footprint files, which can be loaded into the layout cellview.

The Virtuoso custom placer is particularly effective for row-based placement of both MOS devices and standard cells. The Virtuoso custom placer also offers the advantage of allowing the user to define power rails and creates them automatically. The two possible approaches for using the Virtuoso custom placer are

- Run global placement to minimize net lengths and then run a few passes of detailed placement to refine this initial placement (recommended approach)
- Start with an initial placement (seed) and let the Virtuoso custom placer refine this placement using a few passes of detailed placement

The following case studies illustrate some of the device-level floorplanning techniques
.

Floorplan Case Study 1 (Virtuoso XL Layout Editor)

You just finished going through the layout generation process in Virtuoso XL, using the *Design – Gen from Source* command to create pins, instances, and prBoundary. Even though you already had an estimate of the block size and aspect ratio (based on the top-down requirement), you wanted to do a sanity check against the tool estimation. You had selected an utilization factor of 25% and provided the aspect ratio number in the Layout Generation Options form, for generating the prBoundary. You had also turned on automatic folding and chaining options so that the devices are folded and abutted wherever applicable and the complementary pairs are vertically aligned. Because the *Design – Gen from Source* command tries to maintain the schematic topology as much as possible, the devices are already placed relative to the schematic.

1. You first need to instantiate the power pins. Choose the power pins and choose the *Connectivity – Propagate Nets* command. In the Propagate Nets form, set the net names to terminal names to propagate the power pins to the top-level. Use the *Edit – Hierarchy – Flatten* command and the *Preserve Pins* option to preserve the connectivity.

2. Load an existing template file containing pin placement information to place the I/O pins per the top-level spec. The template file was created using the *Design – Save Design to Template* command. Make sure the cellview name in the file and the pin names match the current cellview.

3. To easily view the connectivity, create three separate net classes, one for I/O pins, one for power pins, and one for critical nets using the *Create Class* option in the Show Incomplete Nets form.

4. Place the devices inside the prBoundary. The first check to perform is to see if the gate-level cells/sub-blocks are clustered together. Use the *Connectivity – XL Probe* command to cross-probe the gate-level cells in the schematic, which automatically highlights the corresponding devices in the layout.

5. After making sure you are satisfied with clustering, and making some manual changes, choose the *Connectivity – Show Incomplete Net* command. Choose the net class corresponding to the I/O pins and start moving devices/clusters that hookup to the I/O pins, making sure to move the entire cluster for the gate-level cells/sub-blocks.

6. Move and place the remaining devices to minimize the flight lines for the internal nets. Pay particular attention to the critical nets, which are displayed selectively by choosing the appropriate class under *Show Incomplete Net*. Make sure that the p-devices are close to the power rail, the n-devices are close to the ground rail, and the devices are all aligned in any given row. Also make sure there is enough room between the rows to accommodate routing.

Floorplan Case Study 2 (Virtuoso Custom Placer)

After the initial layout generation and pin placement, you decide to take advantage of the Virtuoso custom placer because a row-based solution seems like the best one in this case.

1. Choose the *Place – Placement Planning* command and choose a row-based, assisted CMOS placement style. In order to get an idea of the placement ahead of time, first estimate the placement-based on minimum number of rows. Looking at the placement statistics, you find that the horizontal and vertical utilization numbers are rather low. You go through a couple more iterations of estimation by selecting a higher horizontal utilization, and then massaging the aspect ratio based on the result.
2. You also modify the folding threshold for the n- and p-devices and go through one or two iterations to improve the vertical utilization. Once satisfied with the placement statistics, you send the design to the automatic placer. Choose the *Group CMOS Pairs* option in the Auto Placer form to ensure clustering of sub-blocks/gate-level cells.
3. Evaluate the placement when it is completed, and save the current state of the design so that you can return to it if needed.
4. Check to see if the devices corresponding to gate-level cells and sub-blocks are clustered together. Use the *Connectivity – XL Probe* command to highlight the devices.
5. Turn on the flight lines selectively for I/O pins, power, and critical nets using the *Connectivity – Show Incomplete Nets* command.
6. Move the devices around to optimize the placement using the flight lines as a guide. Focus on the critical nets and make sure the placement makes sense.
7. Look at the channel space between the rows and the flight lines crossing over to identify crowding, as well as inefficiencies.
8. Manually modify the row height and spacing between rows where applicable. Because you have chosen not to have a very aggressive row utilization, the manual modifications to the placement did not result in any unwarranted growth of the block. You like the changes, and decide to keep the version you just completed.

Placement

Devices should be placed following the basic guidelines for digital or analog components. In addition to the schematic devices, and spare cells should also be placed to account for ECOs that were not already included on the schematic. A typical digital CMOS block may be drawn using one or more rows of n-devices above or below an equivalent row of p-devices. These

blocks can be designed using an unbroken row of transistors in which abutting source/drain connections are made, with channels in between for interconnect.

Figure B-1 Two Transistors Placed without Abutment

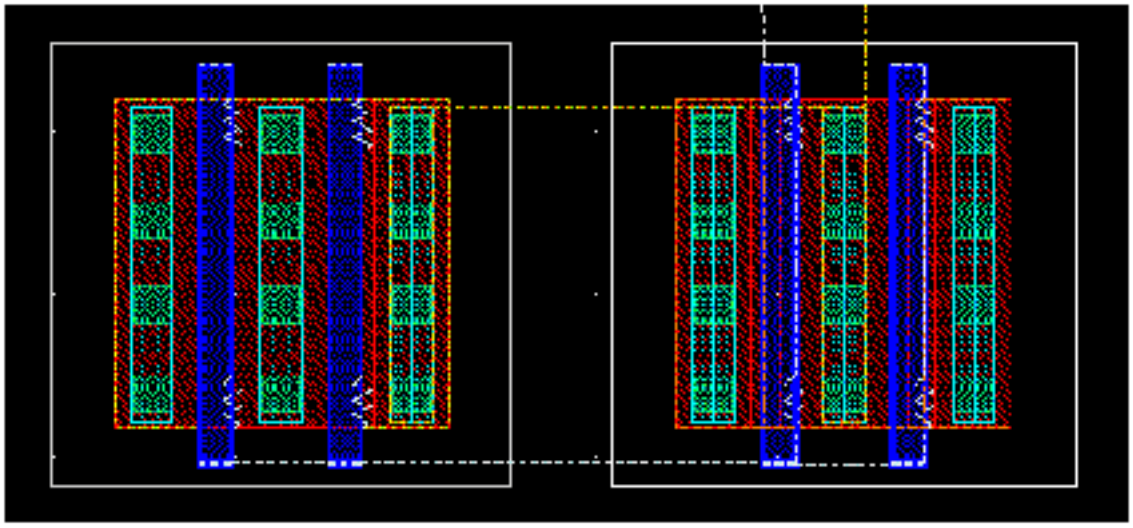
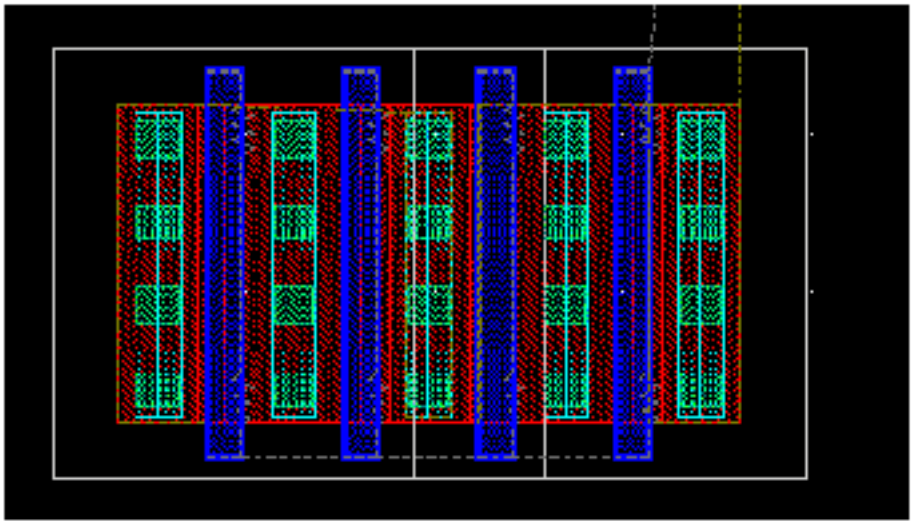


Figure B-2 The Space Savings Obtained with Abutment



There are two main considerations for device placement:

- Placement for short RC delay interconnected route (cells that share the same net should be placed close to one another)
- Placement for better device matching (stack placement and/or common centroid placement)

Placing devices or instances in appropriate locations can determine the performance and block size of the design. You need to make the appropriate trade-off between performance, area, and power. In analog placement, a designer must recognize the critical components and understand all of the random error effects that might occur during fabrication. This includes undercut, gradient or oxide thickness, and thermal gradient effects. Use this information to carefully place the devices for device matching.

Layout of MOS Transistors

Analog MOS transistors have a large aspect ratio. The layout of such elements face problems such as oxide thickness and series resistance at source and drain. Device-matching techniques such as stacked transistors and/or common centroid placement should be followed to reduce these effects.

Stacked transistor placement: With a large aspect ratio, you can choose to use non-straight or serpentine layout transistors to optimize the aspect ratio. This may not be a good idea because the matching of the orthogonal elements is very poor and the contribution to the gate length of the elements at the corners is not well defined. The voltage drop due to the parasitic resistors in the chain is equivalent to an offset that cannot be neglected for precision applications. For this reason, you should avoid using non-straight for analog precision transistors.

To avoid problems and issues using non-straight gate transistor in analog design, a wide straight transistor is commonly split (legged) into a number of parallel and equal transistors arranged in a stack. This leads to a design with more manageable, better-shaped transistors, and with a reduced area. A stacked transistor reduces parasitic capacitance of source and drain to the substrate because of shared source and drain contacts and uses less active area.

- Instead of using one big contact over the source and drain area, use multiple contacts placed at minimum rule spacing to reduce the risk of microfractures. Placing multiple contacts at minimum spacing from the gate (*poly*) also reduces unwanted parasitic resistors.
- The inner gates of the stack see the same boundary (two gates at the two sides), but the gate at the end elements has a different periphery. An unmatched undercut will cause slightly different lengths among the inner gate length and the outside one. For very precise designs, a dummy gate is placed on both ends of the stack to make all gate boundaries result in the same undercut effect and thus improve matching.

- A common application of a stack transistor with two transistors is a differential matched pair. The two transistors are split into equal number of transistors with a common source node and are arranged inter-digitally so the ratio of the two stays the same even through random errors introduced during fabrication. Thus, breaking transistors into multiple fingers and inter-digitizing them in a stack layout improves the gradient effect because the averages of both transistors' characteristics are the same.
- It is good practice to lay out block diagrams or do block planning first before laying out a stack layout, especially a stack composed of multiple transistors. You would have a better understanding when you begin to route. It is a good practice to draw or plan out the inter-digitized blocks first before layout.
- All transistor widths in stack layout should be the same. Depending on the parts into which a transistor is divided, both source and drain or the same terminal can be made available at the ending of the stack. If the different transistors have sources or drains connected to the common node, they can be combined in the stack and share contacts.
- When you combine a number of transistors in a stack layout, sometimes not all positions in the stack are reasonable for some transistors. You must recognize which transistor or transistor pairs are the most critical. The most critical, or transistors with the smallest range of dimension, should be placed in the position of the stack that would most improve matching and other aspects.
- To ensure good matching, all transistors should have the same orientation. The center row is where the critical transistors are planned out. The two sets of transistors are well balanced through the gradient field.

Common centroid placement: In a stack approach for two equal transistors, the matching strategy is to resolve a single dimension gradient effect (X axis or Y axis). For two-

dimensional gradient effect (on both X and Y axes), a common precise practice is to use a common centroid layout structure.

Analog Design Case Study 1

Design a custom analog cell containing inter-digitated as well as cocentric devices, a critical class of nets which require larger widths and an aggressive aspect ratio. The Virtuoso XL layout editor, the Virtuoso custom placer, and the Virtuoso custom router are the tools that you would use. The placer is focused on row-based CMOS designs so your best option is to use the connectivity-driven, constraint-assisted Virtuoso XL environment.

1. After initial layout generation, place the critical devices manually.
2. For the device pair to be inter-digitated, create the devices by picking them from the schematic. Fold them into equal numbers of fingers based on the device legging guidelines. The autoabutment feature alternately places the device fingers which snap in place and share the diffusion.
3. For the cocentric device pair, follow the same approach for generating the devices. Manually arrange them in a common centroid configuration. Use the *Alignment* command to align the horizontal and vertical pairs. By choosing the appropriate layer center or layer edge as the alignment handle, you can automatically apply DRC correct separation as defined in the technology file.
4. Place these sensitive devices inside the block ensuring optimal placement and then lock them using the lock option from the Virtuoso XL menu. Switch to constraint assisted mode to proceed with the rest of the placement, so these devices are not disturbed.
5. Continue with the rest of the layout in constraint assisted mode so you do not accidentally disturb the critical components laid out in a given configuration.

Analog Design Case Study 2

You are working on the layout of a mixer circuit that requires symmetric structures to be created. The procedure you usually follow is to lay out one half of the mixer, create a cell of the half, and then bring in the half as an instance for placement. The problem with this approach was that the connectivity of the other half was incorrect, and another level of hierarchy was introduced within the design. You decide to use the *Clone* command for creating a symmetric or topologically identical structure.

1. Place the devices in the one half of the mixer circuit using alignment, folding, and inter-digitation (chaining of alternate legs) options.
2. Selectively route half of the circuit by exporting a portion of the design to the router with the appropriate constraints (for example, width and layer) and writing out the session file to import the routing back to Virtuoso XL (you could have done the routing manually if the circuit was small enough).
3. Choose the *Create – Clone* command. In the layout cellview, select the portion of the mixer circuit that has been placed and routed. Make sure that all the interconnects are selected as well. This makes the selected layout the (topological) source.
4. In the Cloning form, specify the target schematic and layout. You specify to find target matching for the entire target schematic, and choose *Find Matching Target*s.
5. Choose the targets from the *Unplaced* list box and place them in the layout. You have now created the layout for the other half of the mixer circuit that is identical to the first half. You can now click on the layout window to place the cloned structure in the desired location. Choose the *Flip Vertical* button to mirror the structure along the Y axis to create symmetry.
6. Make sure that there are no overlaps, shorts, or DRC violations resulting from the instantiation of the cloned structure. Choose the *Connectivity – Check Against Source* command to give you an idea of any potential problems.

Layout of Resistors

- For a resistor design, random errors are caused by thermal gradient, undercut or edge effects, or gradient effect due to variation in oxide thickness. These random errors can change resistor values from what a designer expects.
- Most layers in layout can be used to make resistors. In analog design, resistor value can be very big or the ratio of L/W can be quite large, which can cause it to take up a lot of area in the layout. The designer also has to be concerned about the matching accuracy of the resistor due to the random errors mentioned above.

- One of the common practices for laying out a big resistor is to use the serpentine arrangement. In a serpentine arrangement, there are a lot of square corners, which have matching problems. For an accurate design, round corners should be implemented into the bend area or should not be used at all.
- A straight line with low resistance wire should be used to connect all the lumps together. Straight line resistor layout is what designers prefer for precision design because each lump in a layer is the same size and is in the same orientation. All lumps should either be in the X axis or Y axis for thermal gradient matching. Straight line resistor layout is preferred because it gives a high chance for resistor matching.
- The lumps of the resistor body at the edges do not have the undercut effect like the ones in the middle. This boundary-dependent undercut effect can lead to a mismatch in a resistor. To resolve this problem, a dummy lump of the same resistor element is placed on every edge of the resistor lump. This will get the same undercut effect throughout all the resistor lumps.
- For two equal resistor layouts in a gradient effect environment, a designer can use the inter-digitized structure technique. Place each resistor lump interchangeably so the sum of the two resistors is balanced out.
- Noise can cause problems in a resistor, especially in a well resistor. Because the well layer is close to substrate, noise coupling from the substrate can effect the well resistor. A guard ring should be placed around and between the resistor lump.

Layout of Capacitors

- The causes of random errors (or mismatches) in capacitors are commonly the undercut or edge effect and gradient effect (the variation of thickness of the oxide that acts as the dielectric for the capacitor). These random effects cause changes in parameters used to calculate capacitance and should be controlled by using appropriate layout techniques.
- Layers that can generate a large capacitor value for a square unit area are layers that have the shortest distance from one layer plate to the other plate (determined by oxide thickness). For most analog layout environments, the poly2 layer is provided to be used with poly1 for generating a capacitor.
- To minimize the random errors that appear during the fabrication of oxide thickness and gradient effect, and matching of two or more equal capacitor layout, common centroid layout structure is commonly used. Capacitors are usually split into equal parts and placed diagonally across, so that a gradient in the oxide thickness either in the X or Y direction does not affect the capacitor matching. For a very precise capacitor design, even the wire metals are arranged in a way that the wire capacitor is equivalent between the matched capacitor terminal.

- To design a matching or proportional pair of equal capacitors, the perimeter ratio should be the same to account for the undercut effect. In addition, if the plate corner is a 45 degree corner rather than a 90 degree corner, it can improve the undercut effect. For less error, the length and width of the capacitor plate should be equal.
- To protect a capacitor from noise, a guard ring should be placed around the capacitor area. Place a dummy element next to the capacitors that are placed on the edge to improve matching.

Layout of Digital Blocks

- In digital placement, a designer has more freedom in placing the device components. The designer need not be too concerned about device matching from the effect of random edges or gradients. The designer's main concerns are to get the smallest area placement-wise with the shortest interconnects for the least RC delays. To accomplish this, the devices can be placed so that common nets are close together, using the least amount of wire length to route to all the pins to achieve a dense and compact layout.
- In most digital designs, PMOS transistors are placed close to power rails, and NMOS transistors are placed close to ground rails. All components are typically placed to DRC minimum spacing. The more compact the placement, the higher quality the final result. Digital blocks are usually defined with a fixed height and uniform power rails so that each gate can easily abut to another gate.
- Standard gate cell rows are usually used to place digital blocks in a row or column. Most digital layout transistor structure patterns are CMOS structures where the pattern goes vdd, PMOS, NMOS, and gnd. Because most digital standard gate cells have this common pattern, most digital cells are designed to a specific height. With each standard cell, having uniform height, power, and ground rails, a designer can design a compact and dense digital block.
- For both analog and digital blocks, spare cells may be needed for performance improvements during design re-spins, to correct poor results from parasitic simulation, or for an ECO. Place spare cells around the devices that are critical. Adding spare cells can improve turnaround time. It can also prevent a need for redesigning the layout.

The output from this stage is the complete placement for the blocks. The next step is to route nets to all of these device components. The following case studies illustrate some of the device-level placement techniques in an analog mixed-signal environment.

See "Pin Placement" on page 199 for detailed information on the steps to be followed to implement the methodology described in this section. For specific data on assisted manual placement, see "Assisted Manual Placement in the Virtuoso XL Layout Editor" on page 203; or for specific data on using the custom placer, see "Automated Placement using the Virtuoso Custom Placer" on page 208.

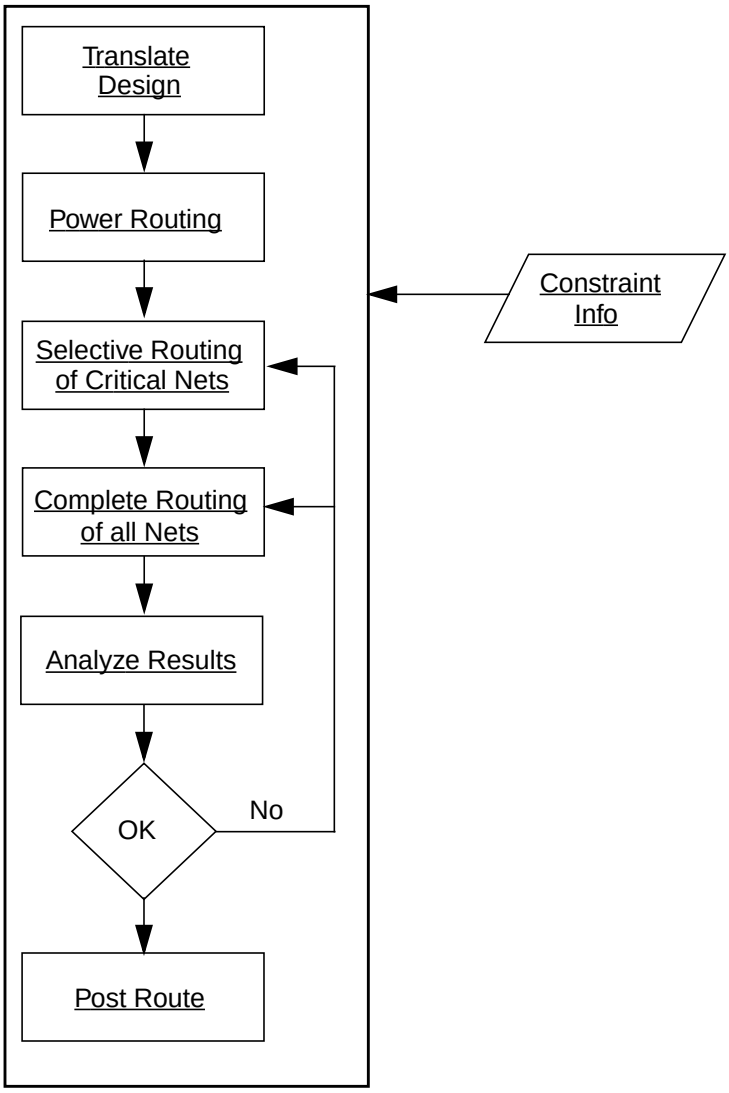
Device-Level Routing

Routing is the next step following the completion of device placement, although placement iterations may often be required during the routing process to achieve an optimal layout that meets all performance requirements. Use the Virtuoso custom router as the routing solution for rapid completion of layouts which are DRC and LVS correct.

The methodology steps for device-level routing are

1. Prepare (and translate) the data for routing
2. Determine which layers are sufficient for routing the block
3. Determine power routing strategy
4. Identify critical nets
5. Identify electrical constraints on the nets
6. Identify nets with any other special requirements (for example, shielding)
7. Complete routing based on above, analyze the results and iterate if required

The following flow diagram illustrates the steps for device-level routing (click on a link to view more detailed information):



Preparation for Routing

Define the Routing Rules

The Cadence DFII library must contain all the connectivity information to run Virtuoso XL as well as design rules for each layer used in the routing environment. Specific Virtuoso custom router setup files may be needed to address special rules not addressed by the layer-based

design rules. For example, a rule that is not specified in the technology file is metal overlap of contact or via in the direction of the route.

Translation Rules File

When Virtuoso XL is started, the layout is linked with the corresponding schematic, establishing the connectivity data for the router. This information is translated to the Virtuoso custom router. Routing layer data, equivalent layers, boundary layers via data and keepout data must all be defined for the router.

Constraints

Constraints are additional rules imposed on specific nets to meet performance requirements. Constraints are added either in the schematic or layout and translated to the router during the router translation step. Constraints can also be added within the router environment using the Virtuoso custom router interface. Generally, constraints should be managed using the Virtuoso Constraint Manager which is available from the CIW. More information on constraints is in "The Virtuoso Constraint Manager" on page 151.

Placement Assessment

Optimal placement is the key to obtaining good routing results. Items to consider:

- For routing the devices, metal layers are recommended rather than other conductive layers such as *poly*. Because *poly* is more resistive, avoid using this layer for routing.
- If the design uses poly, the most important routing objective is to minimize the amount of poly routing. A possible set of routing constraints at device-level would be poly routing cost high or forbidden. This allows metal1 to route in orthogonal mode with free reign. Metal2 is then set to horizontal or vertical. If more metal layers are being used they would then alternate directions. If you use more than two layers of metal, then you have the ability to route over the block at higher levels of the design.
- If the signal on the net is more current dependent, the resistance of the wire affects the signal more than the capacitance of the wire. To reduce the resistance in the wire, make the wire wider.
- If the signal on the net is more voltage dependent, the capacitance of the wire affects the signal more than the resistance of the wire. To reduce the capacitance in the wire, make the wire narrower and increase the spacing.
- When several pins are routed to the same net, a bus route may be required to minimize wire length so that all of the signals can be easily linked to the net.

- Digital block routing usually limits the use of the existing metal layers saving some other metal layers for top-level routing over the block. Depending on the process technology, there may be three or more metal routing layers. For example, for three-layer metal technology, device-level routing is constrained to a single metal routing layer, saving the other two layers for vertical and horizontal routing at the top-level.
- Because routing over analog blocks is commonly not practiced due to noise, routing in an analog block is not constrained to a limited number of metal layers. Use all the existing metal layers in the technology to route an analog block.
- For optimal routing traffic, metal layers are assigned in either the vertical or horizontal direction which avoids shorting and routing traffic. In most routing practices, an odd numbered metal layer is assigned for horizontal routing, and an even numbered metal layer is assigned for vertical routing.
- Appropriate measures should be taken for reduction of noise injection due to capacitive coupling. To prevent noise and crosstalk in sensitive and critical signals, keep the lines as short as possible. Decouple by using a large spacing between the lines and/or shield the lines.

Implementation Details

Refer to "Translate Design Detailed Instructions" on page 215 for detailed information on these steps.

Interactive Routing

The Virtuoso Interactive Wire Editor is used within the Virtuoso XL environment and gives you interactive routing and editing features which support online DRC checking through technology file process rules. Refer to "Interactive Routing capabilities" on page 217 for detailed information on these steps.

Power Routing

The first step in routing the design is to lay down the power trunks. Global signals like vcc and gnd will already be defined as trunks by the Virtuoso custom placer as part of the placement step. It is possible to route these signals without using the power router but the trunk attribute must be removed.

Refer to "Power Routing Technical Details" on page 227 for detailed information on these steps.

Automated Routing

Automated routing is a two-step process. The first step is to route the critical nets. The second is to use the autorout (that is, to route the rest of the nets).

Shut down the router if you need to go back into Virtuoso XL and regenerate the design.

Critical Net Routing

There are some nets that must have special attention to achieve performance and reliability goals. These critical nets must be identified and routed first. Routing can be done manually

as preroutes or the critical nets may be constrained. Case studies have been included to demonstrate some of the techniques.

Shielding of Differential Pair Nets Case Study

Your company is in a 90% analog full-custom layout environment. They use and define multiple sets of differential pairs for routing. As a postroute process, they are also required to shield these differential pairs. All shielding options will be used but primarily they will use parallel shields.

You have tried several approaches, but with limited success. You contact the Cadence Hotline and receive the following solutions. Use the following Virtuoso custom router command sequence to allow the shielding of differential pairs:

1. Choose the nets to be defined as a net pair.
2. Choose *Define – Net Pair – Selected* and specify a gap.
3. Route the selected net pair.
4. Choose *Define – Class – Selected.*
5. Choose *Define – Net Pair – Define/Forget By List.*
6. Choose and forget the defined net pair.
7. Choose *Rules – Class – Width/Clearance* (optional).
8. Choose *Rules – Class – Wiring* (optional).
9. Choose *Rules – Class – Shielding.*

You noticed that routed differential pairs did not shield when you forgot the defined net pair followed by defining them as a class. You must define the class (selected) first, followed by the forget net pair.

Controlling Nets and Subnets Using Pseudo-Pins Case Study

You have some critical nets that you not only want routed separately, but want to control the formation of the net and subnets. You do not want to necessarily preroute the wires because the block sizes and locations will be changing as the project progresses. You need to have the main net routed as a trunk, and you need to control the widths of the subnets that branch off to specific devices. After some trial and error, you refine the steps down to the following:

1. Prior to running the Virtuoso custom router, create a `.do` file that contains the subnet list with desired widths. A subnet is called a From-To in the Virtuoso custom router.
2. Start the Virtuoso custom router.
3. Turn *Trunk* Mode on.
4. Choose the nets to be routed as a trunk.
5. Load the `.do` subnet file created in step 1.
6. Route the selected nets.
7. Turn *Trunk Mode* off.

Using this approach, you can easily reuse the subnet `.do` file during design iterations.

Normal Nets Routing

Once the power and critical nets have been routed, the rest of the nets can be routed with an emphasis on minimizing routes on specified layers. For example, poly routing needs to be minimized for performance reasons. This requires that *metal1* and *metal2* be used first if possible and that *poly* be used last and with a higher cost in the routing scheme.

Refer to "Routing Steps Technical Details" on page 231 for detailed information on these steps.

Strapping of gates, drains, and sources in MOS devices, using predefined rails (Virtuoso XL Layout Editor and Virtuoso Custom Router) Case Study

A layout has been generated with a number of folded and chained/inter-digitated MOS devices and some fingered devices. Strap the gates of these devices in the layout, using specified metal layer and width, with minimal poly routing. Also strap the drains/sources of the inter-digitated differential pairs.

1. After choosing the appropriate metal layers based on performance requirement, create the rails in Virtuoso XL for strapping the gates of the fingered/folded devices and for strapping the source and drain terminals. Choose *Create – Path* to draw the rails and enter the desired width in the Create Path form. Specify a net name.
2. You don't want to create any additional pins to place on these rails in order to assign them connectivity. Some of the nets are internal and do not connect to any I/O pins.
3. Choose *Options – Layout Editor* and turn on the *Sticky Net* option. The *Sticky Net* option assigns connectivity to these rails by using the net name assigned in the Create Path form. Use the *Connectivity – XL Probe* command to highlight the particular net and to cross-probe between the schematic and the layout in order to make sure that connectivity is correctly assigned.
4. Align the rails for strapping the gates to the appropriate groups of instances with minimal spacing between the *poly* terminal and the rails. Place the rails that are meant for strapping the drains/sources over and across the abutted/folded/inter-digitated devices that need to be strapped. Use the stretch option in partial select mode to size the rails.
5. Export the design to the Virtuoso custom router. Using a `.do` file at startup, make sure that stacked vias are allowed for all layers, in order to allow the *poly* terminal of the gates or the *metal1* strip for the drains/sources to hook all the way up to the higher level metals that the rails are drawn on.
6. Turn on pin-to-trunk routing using the power router to strap the gates of the legged devices to the predefined rails.
7. To strap the drains and sources of the folded/inter-digitated devices to predefined rails that run across and overlap the devices, use the *Via on Mesh* routing option of the power router, after providing the source and target layer information.
8. Leave the *Via array* option blank to let the router extend the via array across the width of the rails.

Routing Analysis

You may not always achieve 100% route completion on the first pass or even if you achieve a complete route you may want to further optimize the layout. Reasons for less than 100% completion or non-optimal results include router failures to find a path due to poor placement/ high congestion, a blocked pin, and overconstrained nets.

If the Virtuoso custom router cannot complete routing certain nets you will need to adjust your placement to accommodate these nets. Use the *Place* mode within the Virtuoso custom router to adjust the placement or adjust the design. The Message Passing System (MPS) keeps the design in the Virtuoso custom router synchronized with any edits created in Virtuoso XL. One advantage of using the Virtuoso custom router is the push and shove capability that allows your placement edits to remain design rule correct. Virtuoso XL does not provide this capability.

Refer to "Routing Analysis Technical Details" on page 236 for detailed information on these steps.

Analog Design Case Study (Virtuoso Custom Router)

You have the task of designing a custom analog cell containing inter-digitated as well as cocentric devices. The critical class of nets require larger widths and an aggressive aspect ratio.

1. Before leaving the Virtuoso XL tool, create a class of nets in Virtuoso XL called *critical*. Then place a larger width on the class of nets. Preroute the power rails in Virtuoso XL by placing a power template in the cell and flattening the template.
2. Next, start the Virtuoso custom router to route the devices. The components and prerouted wires are automatically protected in the Virtuoso custom router. The router will not move any component that is protected. Interactively route the critical nets class in the Virtuoso custom router and tweak the routing. The width of the critical nets had been preset, so they are routed with the proper width.
3. During interactive routing, you realize that the placement of several devices needs to be adjusted. Because you turned on the Virtuoso custom router from within Virtuoso XL from the *Route* menu, the Virtuoso XL and Virtuoso custom router environments are synchronized. Any placement edits made in the Virtuoso XL window are automatically reflected in the Virtuoso custom router window (the underlying technology is MPS).
4. With the Virtuoso XL placement adjustments finished, complete routing the few remaining nets using the Virtuoso custom router *Route* command. The routing results are reflected back in the Virtuoso XL layout editor window.
5. Finally, run DRC and LVS checks from the Virtuoso XL window. Some of the placement adjustments in Virtuoso XL resulted in a well spacing error. Correct the spacing and clean up the nets in Virtuoso XL. The connectivity-driven environment maintains that the schematic is consistent with the layout throughout the entire edit process. The LVS check is 100% matched on the first pass.

Shielding Case Study (Virtuoso XL Layout Editor)

You finished importing the design from the Virtuoso custom router back to Virtuoso XL.You subsequently realize that one particular net needed to be shielded. Instead of going back to the Virtuoso custom router, setting the appropriate constraint, routing it, and importing it back again to Virtuoso XL, you decide to fix the net in Virtuoso XL. You also want to try out the option for ROD-based multipart path creation.

1. Choose the *Edit – Create Multipart Path* command.
2. In the LSW, select the *metal2* drawing for the net. In the Create Multipart Path form, also enter the required width for the net. Provide the connectivity information for the net by changing the *Connectivity* option to *Net*, and then providing the net name.
3. Choose *subpart*, and in the ROD Subpart form, select *Offset Subpath*. Change the layer to *well* and changing the width information for the shield. Also, set the separation distance of the shield from the main net, and change the justification to *left*. The net name is set to *informGND*. Click *Add* to add the subpart information.
4. Because the net needs to be fully shielded (shielded on both sides), repeat step 3 with everything identical except for justification, which is *right*.
5. You can create a fully shielded net in the layout.

The Virtuoso Constraint Manager

Constraints

The process of adding constraints to a design to meet performance requirements is a normal part of the device-level design process. Constraints are not design rules that can be checked by a DRC routine but rather are additional requirements such as device matching, wider net widths, and net shielding, which are unique to each circuit and are not process dependent. These constraints can be added initially when certain performance criteria are known and understood or after several parasitic resimulation iterations when additional layout characteristics related to parasitic capacitance and resistance are better understood.

Typically, in non-automated flows, this involves verbal or written information that the circuit designer passes to the layout designer. Each company or group might have its own unique way of communicating these constraints. As the flow becomes more automated, these constraints need to become more formalized and stored electronically to allow tools to interpret and process the constraints. Constraints can pertain to both routing and placement. These constraints can be added at any point in the design flow from schematic to layout, but in general the best practice is to include the constraints in the schematic level so that when the cell is reused the new layout will retain the performance constraints from the previous design. Automating the constraint implementation can produce consistently good layout results once the implementation details have been captured and proven.

Implementation Considerations

The technology that supports electronic capture, storage and implementation of design constraints includes the Virtuoso Schematic Composer, Virtuoso Constraint Manager, Virtuoso XL Layout Editor, Virtuoso Custom Placer, and the Virtuoso Custom Router. Currently there is no postlayout tool to check that all constraints have been met. The Virtuoso custom router will notify you of constraints that have been relaxed. Constraints that are not handled by the Virtuoso custom router are not checked for correctness and must be manually checked in the Virtuoso Constraint Manager.

To create constraints, open a schematic (or layout) view and choose *Tools – Constraint Manager* from the CIW. Constraints for nets and components can be added, viewed, and deleted using the constraint manager.

- Constraints defined at the schematic level are inherited to layout views with which a cellview pair relationship exist
- Constraints placed in the layout do not transfer back to the schematic

- Selected constraints, such as constraints inherited from the schematic, can be controlled by activating them off or on as desired

Specific constraints need to be tested in isolation to evaluate circuit performance. By entering the constraints and using the tools to quickly reroute the design, many iterations can be used to improve the circuit very quickly. The types of constraints include

- Net-based (for example, set no-cross rules, shield nets, width > minimum)
- Distance (for example, keep devices a required distance from another device)
- Alignment (for example, align devices in X and Y)
- Grouping (for example, keep devices together - one moves then all move)
- Symmetry (for example, keep devices symmetrical independent of orientation)
- Fixed (for example, lock components to a fixed location)

Placement-Based Constraints

Placement-based constraints include all of the constraint types except net-based constraints and are related to devices and components. These include

- Distance – max or min distances for components
- Alignment – vertical or horizontal
- Grouping – components can be grouped to maintain a relative relationship or to specify an area that contains only those components (using a fence)
- Symmetry – components can be defined to be symmetrical about an axis or two components can be symmetric with respect to each other
- Fixed – to lock components into a specific location, a fixed orientation, or a fixed direction along the X or Y axis

These constraints drive Virtuoso XL and the Virtuoso custom placer. Placement constraints can be used to restrict the placement of the devices to fixed or relative positions on the layout. For example, a common centroid device configuration can be defined using a symmetry pair constraint. These constraints allow restricted placement of sensitive or critical devices, as well as to achieve a desired floorplan.

Additional Considerations for Placement Constraints

Virtuoso XL enforces constraints that are already met in constraint-assisted mode. For example, setting an alignment constraint does not cause Virtuoso XL to automatically snap

components into an aligned position. If you place the components so they are aligned, Virtuoso XL will enforce the constraint by not permitting you to move the components out of alignment. If a distance constraint is set, Virtuoso XL will not let you violate the constraint once it is met. Constraints created in Virtuoso XL are passed to the Virtuoso custom placer and will be obeyed as long as the placer is not overconstrained to the point of not being able to complete the placement. For more information on how constraints interact with the placer constraints, refer to the Virtuoso Custom Placer chapter in the *Virtuoso XL Layout Editor User Guide*.

Net-Based Constraints

Net-based constraints drive the custom router. The constraints can be defined for specific nets or net classes. A net class is a group of nets that is given a name. Constraints are automatically converted to a `.do` file, *`cellname`* `Constraints.do` and translated to the custom router.

Net-based constraints can also be added in the Virtuoso custom router. These constraints are not stored in the DFII database and must be manually saved if they are to be reused. The recommended methodology is that all constraints that are to be reused and that need to be available in DFII be entered prior to starting the Virtuoso custom router.

Additional Information on Net-Based Constraints

For information on how the Virtuoso custom router uses constraints, refer to the *IC Shape-Based Technology Chip Assembly User Guide*. The Virtuoso custom router is a constraint driven router that is driven by *routing rules*. You can read more about these rules in the Setting Routing Rules chapter of the *IC Shape-Based Technology Chip Assembly User Guide*.

Limitations of Folded Devices

In the case where a single device gets folded into multiple instances in the layout view, the new instances will inherit all of the original instance properties. This can cause conflicting constraints. You will need to resolve any conflicts that may arise from this operation. All such cases are reported in the CIW.

Physical Verification

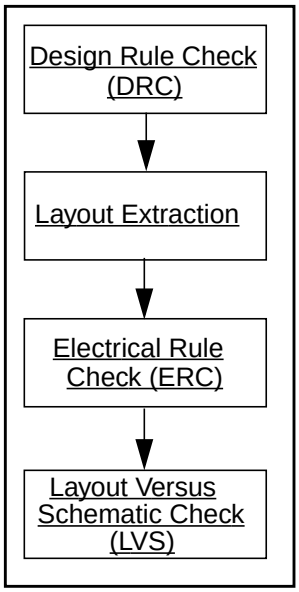
The Verification Process

There are several required steps to formally verify that the layout matches the schematic, is manufacturable, and will work properly. The physical verification process checks the layout against the original design and against the layout rules that provide specifications to ensure manufacturability.

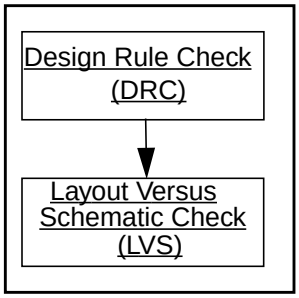
The ERC and LVS extraction netlist does not usually need to contain parasitic information. But for parasitic simulation, a netlist must be extracted with parasitic resistance and capacitance information for all layer characteristics. While this step is not a verification step in and of itself, it is necessary for later verification, and it does check for netlisting syntax.

The following flow diagram illustrates the steps for the physical verification process (click on a link to view more detailed information):

Diva Physical Verification



Assura Physical Verification



The Design Rule Check (DRC)

The DRC is the first verification task and should be run during layout generation and after layout is complete. DRC checks the layout to see if layers are being placed according to the rules specified by the process manufacturer. Although there is no real dependency between DRC and extraction, it is important to fix all DRC errors prior to proceeding with the remaining verification steps. DRC correctness is the most basic requirement for manufacturability and correcting DRC errors upfront reduces layout iterations. All of the steps shown below are contained in the Diva® Interactive Design Rule Checker tool and in the Assura™ Design Rule Checker tool.

Design Rule Check Basics

The DRC is a part of layout verification. The DRC checks layout designs to see if they satisfy the constraints imposed by the fabrication process for a selected technology.

No matter how conscious a designer is of the design rules when performing the layout, they are often overlooked. In general, before designs can be manufactured, they must pass DRC. The quality of the DRC depends on the DRC rule check.

A quality DRC rule deck contains several rule checks such as width, spacing, dog bone, overlap, notch, grid, and antennas. Different fabrications have different DRC rules, even if the technology is the same. In some cases, there are several decks to permit isolated checking. All of the decks should be run.

Sometimes when a designer routes a design using a layout tool, the wire does not appear to be shorted to another wire or opened, but after the design is fabricated, the layers can be shorted to another layer or an unintentional open circuit can show up. Passing a DRC check guarantees the following:

- The layout will not be shorted or open because drawn layers are too close or the layer path width is too thin
- Each transistor type is assembled correctly from a set of layers
- The layout that is output will be masked and etched accordingly

During placement or routing, run several iterations of DRC checks. Don’t wait for the entire placement or routing to be completed, because one error can create a domino effect for the whole design depending on the location of the error and extent of the fix.

A designer may need to readjust other cells or routed wires to correct that particular cell. The sections that follow outline DRC checks that must be performed during various stages of the design.

1. **DRC on Block Plan**: This step uses the DRC verification check to verify that the power plan, guard rings, and wells are DRC clean before transistor placement proceeds.
2. **DRC on Placement**: This step checks that device instances have been placed in accordance with design rules, to ensure that all the layers that make up the devices (poly, diffusion, contact, and metal) do not violate DRC rules, and that instance spacings are DRC correct.
3. **DRC on Route**: This step ensures the DRC correctness of the routing. A DRC error can lead to a misconnected, short, or open circuit.
4. **Other DRC checks**: Additional DRC checks can determine such things as overall layout density, which can affect yield.

Design Rule Check Technical Details

You can use the two different tools to perform DRC checks. Specific technical details are described in the sections below.

- Diva Design Rule Check
- Assura Design Rule Check
- Common Design Rule Check Issues

Diva Design Rule Check

1. The input to DRC can be a layout with all devices placed and hooked up, including power/ground rails, guard rings, and wells. However, DRC should also be run on placement alone prior to proceeding with routing.
2. Diva DRC can be run using
   - Virtuoso XL menu: the *DRC* command is under *Verify* in the design window.
   - Batch mode in the CIW: `ivDRC`
   - Batch mode in UNIX:
     `ivVerify drc [-h] [-l libname] cellname viewname`
   - Remote verification in batch mode: Jobs are run on a remote server
3. Options for running DRC include:
   - Selecting a checking method: The checking method could be either flat (checks the entire hierarchy as though it were flat), hierarchical (checks each cell in the design's hierarchy one time and recognizes repeated patterns so that these patterns are checked only once), and hierarchical without optimization (checks each cell in the design's hierarchy without pattern recognition, so all patterns are checked).
   - Selecting a checking limit: This could be full (checks all data), incremental (checks only the modified data), or by area (checks the area defined by a rectangle).
   - Setting switch names: This can limit design rule checking of specific layers or types of data.
   - Including or excluding cells for checking: This is done through `ivIncludeValue` property in the cell to be included or excluded.
   - Joining nets with the same name: This option will assume they are to be electrically connected.

- Selecting a rules file and rules library: The rules file typically resides in the technology library and the default name is `divaDRC.rul`. Other DRC rules files may also be supplied for specific purposes.

4. Once the right switches and rule deck have been selected, click *OK* or *Apply* to run DRC. The result of the DRC run appears in the CIW. The window can be scrolled as necessary to see all the errors. The summary (number of errors and number of warnings) will appear at the end.

5. Errors, if any, can be located with the *Verify – Markers* commands. Selecting *Explain* opens the Explain Text window, showing messages for the DRC errors that are pointed to. *Find* opens the Find Marker form, which has options for displaying DRC errors. *Edit – Search* shows messages for errors found by the search filter defined by the user. The `drcWhy` property and its value can be used to find specific error types.

6. To create an output file for the DRC results (for future reference), DRC can be run at the UNIX prompt (as mentioned earlier) and the result directed to a UNIX file, for example

   `ivVerify drc [-h|-f] [-l` *`libname`*`] `*`cellname viewname > outputfile`*

   where h is the option for hierarchical check and f is for flat check.

   Note: There are certain discrepancies between the hierarchical check and the flat check options in DRC. The report generated by hierarchical check can show violations even though the design is DRC clean.

For more information, see the *Diva Reference Manual*.

Assura Design Rule Check

1. The input to DRC can be a layout with all devices placed and hooked up, including power/ground rails, guard rings, and wells. However, DRC should also be run on placement alone prior to proceeding with routing.

2. The Assura DRC can be run using batch mode in UNIX.

3. To run in batch mode from a UNIX command line,

   a. Create an RSF file (run-specific file) for the design. Usually an existing file can be copied from another job, or a template file can be edited so that it specifies the design library, cell, and view, and other run-specific details. To run DRC on a DFII layout and schematic cell called `mosTest` located in a library called `test`, the file might look like this:

```
;DFII Data
?inputLayout (“DFII” “test”)
?cellName “mosTest”
?viewName “layout”
?rulesFile “assuraDRC.rul”
```

```
?runName "mosTest"
?workingDirectory "assuraRunDir"
```

b. From the UNIX command line, type

```
assura assuraDRC.rsf
```

A directory called `assuraRunDir` will be created by the Assura DRC and will contain the Assura files from this run.

c. Once the batch job is complete, access the results by choosing *Assura – Open Run*, enter the path to the run results, and click *OK*. The Error Layer Window (ELW) appears with the DRC errors found by that run. Keep the ELW window open.

4. Information for running an interactive Assura DRC is given below:

5. Using the form that appears when you choose *Assura – Setup – DRC* is optional, and it contains a mechanism for users to customize settings which ultimately end up in the RSF file. A user may use this form to customize the RSF, or instead, use an include mechanism from the Run DRC form to include the same and/or different RSF options. Which options and settings that should be used depends on the rules that come from the foundry.

6. The *Assura – Run DRC* command brings up a form with several fields and options.

Note: At the top of the form are the *Load State* and *Save State* commands. One may load a state file, which seeds the entire form with saved settings. Once completed by a user, the form may be saved to allow easy retrieval of the same settings.

7. Choose a checking limit. A full check is for the entire design. A by area check will check a user specified rectangular area of the design.

a. Run directory: The default is ".", which is the working directory where the software was started. A directory name can be added under which all Assura files will be created (for example, . ./assura/DRC) if desired.

Note: This directory must be created manually from UNIX. The *run* command will not create this directory.

b. Run name: The default is the same as the cell name, which can be overridden if desired.

c. RSF name: This is the output filename that the form, once executed, will write to. The default is `cellname.rsf`.

d. Specify the DRC rules: There are two primary methods available to choose DRC rules.

The first is to use *Assura – Technology* to create a technology library that is then selected in the Run DRC form. The second and more common method is to not use the technology method by leaving the *Override Technology Selection* choice selected, which then allows the user to specify a specific path in the *Rules File* section.

e. Set Switch names: After a rule file is specified, click on *Set Switches* for a list of choices that the rules define (if any). Whether to select switches or not depends on instructions for using rules, or as part of rules obtained from your foundry.

f. RSF Include: If desired, one may include a `user.rsf` file, which will be inserted at the top of the complete RSF file that the UI creates and submits to the Assura DRC. This RSF Include file is usually of the form

```
avParameters(
?ignoreGetAngledEdgeDifference t
?joinPins t
?runName ‘run_name’ ;this line is overriden by the form rsf
;?runName ‘<run_name>’ ;this line can’t be used due to the  illegal value.
)
```

A foundry-supplied RSF file could also be specified. The `runName.rsf` file is assembled by first including your `user.rsf` file, followed by the RSF from the form settings. However, if any duplicate options occur in the `user.rsf` file compared to what the form will generate, the form values will override the entries in the included RSF file. After submitting the DRC form, examine the `runName.rsf` file to understand exactly what `user.rsf` options will be overridden.

g. Template file: A template file is generated by using the form that appears when you choose *Assura – Setup – DRC*, which can then be included via this field on the Run DRC form.

h. Options: Choose the *Options* command to bring up a form with a list of run options. Whether each command is editable or usable depends on what settings have been chosen in the *Assura – Setup – DRC* form. Regardless of the form settings, the *Use* button must be clicked for that option to appear in the RSF file. Also, in the *Grid Size* field, enter the manufacturing grid. Click *OK* or *Apply*.

8. Run the DRC: Once the right switches, rule deck, and other settings have been selected, click *OK* or *Apply*. A Progress Form appears. It is recommended to click on *Watch Log File* to observe progress of the DRC job. Regardless of whether the log file is monitored, or whether or not the Progress Form is closed, a window will pop up when the job completes stating that the job has finished. The pop-up menu can be used to view the results at that time. Clicking *Yes* brings up the Assura DRC Error Report (a summary of all the violations in the design) and the Error Layer Window (ELW).

9. If the ELW is not already open, choose the *Assura – Open ELW* command. Leave this form open until completely finished with the Assura DRC results.

10. The left side of the ELW lists which errors are violated, and how many for each rule. Click on any error. Click on the right or left arrows to zoom in to each error. Also observe that one or more cell names appear in the right side of the ELW, along with an error count next to each cell. These are the actual cells which contain the errors. Choose a cell name. Notice the *Open Report* command is then selectable. Choosing it will cause a text report of those errors to appear, which includes hyperlinks to specific coordinates for each error. These hyperlinks or the left-right arrows can be used to zoom in to each error.

11. Assura DRC rules are written in Assura syntax. If the user desires to submit Diva DRC rules to Assura, in most cases it will work without any changes. The Assura DRC can translate most Diva syntax into Assura syntax. Sometimes there are differences, and the CIW output often indicates which rules are problems and often what to do about them. Because of differences in Diva and Assura capabilities, The message might say, for example, to write a Diva rule differently, or to override and ignore the difference. Include a command option in the RSF file, such as

```
?ignoreGetAngledEdgeDifference t ;see the RSF example above
```

Note: To use Diva DRC rules consistently, it is suggested that they be converted to Assura syntax by setting *?compileOnly t* in the RSF file. That will cause Assura to translate the Diva rules (for example, `divaDRC.rul`) and write them out to a file in Assura syntax (for example, `divaDRC.rul.av`). This file can then be examined, modified, and enhanced to take full advantage of all the features that the Assura tool offers.

It is strongly suggested to refer directly to the Assura documentation, and if possible, attend an *Assura Verification* training course. While similar to Diva, the Assura verification tool still has differences in UI and with the methods used to debug with a true hierarchical verification tool.

Common Design Rule Check Issues

- With most rule decks, it is necessary to provide substrate/well taps at an interval governed by the process technology. If the pcells have a parameter for connecting the source to the body, then that can be turned on in the layout as appropriate. Otherwise taps should be inserted manually in the layout as per guidelines/DRC rules.

  Note: Contacts for hooking up wells and substrates to *metal1* should be used. These contacts should be placed in the design at required intervals in the wells (for p devices) and substrate (for n devices) and should be hooked to power or ground, as appropriate. Similar approach should also be taken for 4 terminal devices whose bulk contacts need to be hooked to a separate voltage source (dummy well/substrate).

- If running DRC on placement alone, judgement should be used to ignore violations due to the fact that instance/pins are not connected (for example, minimum M1 or M2 area violation for the pins).

- Certain DRC rules may not be understood by the Virtuoso custom router which gets its rules from the technology file. In particular, it is a good idea to turn on *Same Net* check and *Corner to Corner* check in the Virtuoso custom router (*Rules – Check Rules – Setup*). Also, all notches should be removed in the Virtuoso custom router (*Autoroute – Post Route – Remove Notches*) if the DRC deck check for notches.
- Some DRC checks provided in the rule deck may not be appropriate for device-level blocks or cells and would be more relevant in a hierarchical design. Some of these violations (for example, certain antenna errors) would go away when run at a higher level.

Netlist Extraction for Electrical Rule Check and Layout Versus Schematic

The purpose of this step is to extract the netlist layout without parasitics for an electrical rule check (ERC) and a Layout versus Schematic check (LVS). This step is tool specific and not required as part of the Assura flow. Use the extractor to extract only devices and their parameters and connectivity data from the layout. Because this netlist will be used either to compare the layout against the schematic or for checking the electrical rules, parasitic information should not be included in the extraction procedures because this would cause false errors resulting from the parasitic components. If the schematic and layout have similar parasitic structures because the engineer was trying to model these values, then it would make sense to run LVS with a parasitic netlist. All of these checks are contained in the Diva LVS tool.

Layout Extraction Technical Details

The following procedure is specific to the Diva flow. For information on the Assura flow, go to Assura Layout Versus Schematic on page 168.

1. The input to the Diva layout extraction tool is layout with all devices/sub-blocks placed and hooked up. It is recommended that the layout be DRC clean.
2. Extraction can be run using
   - The Design Framework menu: Choose the *Extract – Verify* command from the design window.
   - Batch mode in the CIW: `ivExtract`
   - Batch mode in UNIX:

     ```
     ivVerify extract [-h] [-l libname] cellname viewname
     ```

   - Remote verification in batch mode: Jobs are run on a remote server

   Options for running extraction include

- Specifying an extraction method: The extraction method could be either flat (creates a single-level extracted view), hierarchical (extracts each cell in the design's hierarchy and creates a single extracted view for each cell, in which the other cell instances appear only as device instances), and macro (creates a flat-level extracted view and stops the extraction at the cell level for all cells designated as macro cells).
- Joining nets with same name: This would assume them to be electrically connected.
- Setting switch names: Switches can be set for extracting with or without parasitics (the options being *Cparasitics*, *Rparasitics* and *lvsonly*).
- Setting the inclusion limit.
- Specifying a rules file and rules library: The rules file typically resides in the technology library and the default name is `divaEXT.rul`.
- Specifying view names: View names could be specified for extracted view or excell view (a modified version of the extracted view of a cell containing pin and boundary information).

3. Once the correct switches have been selected, click *OK* or *Apply* to run extraction. The results appear in the CIW. The summary (number of errors and number of warnings) appears at the end.

4. Errors, if any, can be located with commands under *Verify – Markers*. When *Explain* is selected, the Explain Text window opens, showing messages for the extract errors that are pointed to. *Find* opens the Find Marker form, which can display extract errors. *Edit – Search* shows messages for errors found by the search filter defined by the user. The `extractWhy` property and its values can be used to find specific error types.

Note: As mentioned earlier, it is not necessary to have an extracted netlist with parasitics (that is, to do parasitic extraction) for LVS. If the design is not too large, it might be a good idea to extract with parasitics prior to running LVS. This is likely to cause false errors do to phantom components generated by the parasitic extraction. These can be noted and resolved manually. However, if the design is large and requires many iterations, it might be time consuming to extract netlist with parasitics and to resolve the false errors.

5. If it is desired to have an output file for the DRC results (for future reference), DRC can be run at the UNIX prompt (as mentioned earlier) and the result directed to a UNIX file, for example

```
ivVerify extract [-h|-f] [-l libname] cellname viewname > outputfile
```

Note: In order to run extraction in batch mode in UNIX, it is essential to close the target cellviews.

Common Extraction Issues

If there are multiple pins for a net in the design, they may not be directly connected (for example, devices may hookup to two separate power or ground rails but the rails may not be connected at the block/cell level, and instead connect to the grid at the next level of hierarchy). The extractor may flag this as a violation. Either these nets should be hooked up at the device/block level itself or *Joining Nets With Same Name* should be turned on in the extraction form.

For more information, see the *Diva Reference Manual*.

The Electrical Rule Check (ERC) – Optional

When the layout work is complete and passes DRC, the electrical rule check (ERC) is run to check for open and short circuits. The ERC check is primarily for global checking, such as a power connection check, and is used for the following:

- To check the electrical syntax of the layout
- To perform a fast circuit analysis

The electrical rule checker (ERC) is a quick and time saving analysis that checks the layout for electrical rules. ERC is similar to a simulator checker, checking the circuit netlist for errors such as undriven nodes, unused transistors, floating nodes, shorted nodes, and power/ground nodes. There is no ERC check in the Assura flow, however many of the checks can be performed as part of the LVS check. The design of the Assura tool permits isolation of the two error types without the confusion normally associated with skipping this test.

ERC should be run prior to LVS. If there are errors in the ERC, there are bound to be errors in the LVS check. It is best to pass ERC prior to running an LVS check, leaving other mismatches that ERC is not able to detect for the LVS check. ERC errors, if not fixed, can cause LVS to give some false extra errors. Hence, it is best to run ERC prior to LVS, especially when dealing with large-scale designs. LVS can check what ERC can, but it would be difficult to troubleshoot the errors, especially with false errors.

The quality of the ERC depends on the ERC rules. To ensure a successful ERC, verify that your rules are comprehensive and complete.

Electrical Rule Check Technical Details

The following procedure is specific to the Diva flow. For information on running ERC in the Assura flow, go to Assura Layout Versus Schematic on page 168.

1. The input to this step is an extracted layout with all devices/sub-blocks placed and hooked up. It is recommended that the layout be DRC clean. It is not necessary to have parasitic information in the extracted netlist for the purpose of running Diva ERC.
2. The *ERC* command can be run using
   - The Design Framework menu: Choose the *Extract – Verify* command from the design window.
   - Batch mode in the CIW: `ivERC`
   - Batch mode in UNIX:

     `ERC [-h] [-l libname] cellName viewName`
   - Remote verification in batch mode: Jobs are run on a remote server.

   Options for running ERC include
   - Run directory name: The run directory will contain necessary files such as `si.env` (environmental control), netlist (the netlist created for ERC), and `probe.err` (the text file containing a list of errors).
   - Generating new netlist: When options are changed for subsequent runs and cellviews have not been edited, there is no need to create a new netlist during the next run and this feature can be turned off.
   - Defining the cellview: Three methods can be used to choose the cellview, point to the cellview window, point to the cellview name in the library browser, or enter the name of the cellview in the form.
   - Selecting a rules file and rules library.
   - Job priority.
3. Once the correct switches have been selected, click *OK* or *Apply* to run extraction. The results appear in the CIW. The summary (number of errors and number of warnings) appears at the bottom.
4. Errors, if any, can be located with commands under *Verify – Markers*. When *Explain* is selected, the Explain Text window opens, showing messages for ERC errors that are pointed to. *Find* opens the Find Marker form, which has options for displaying ERC errors. *Edit – Search* shows messages for errors found by the search filter defined by the user.

   Note: Electrical checking locates basic circuit errors, such as floating lines, lines with only one connection, floating devices, and so on. However, because these errors do get caught in the subsequent verification steps, ERC is not a mandatory verification step.

For more information, see the *Diva Reference Manual*.

Layout Versus Schematic (LVS)

At the layout versus schematic (LVS) stage, the layout implementation is compared to the source schematic. This is one of the main verification steps that determines if the layout has the same circuitry as the schematic. The LVS step provides the designer with feedback as to whether or not the layout matches the schematic.

LVS requires (or generates) an extracted netlist and compares the following areas:

- Connectivity of layout and schematic
- Parameter values between schematic and layout
- Number of components between schematic and layout

The LVS check is a logical verification between two netlists. One netlist is extracted from the schematic and the other is extracted from the layout. LVS compares the two netlists to see if they are the same. LVS can compare at the instance or device-level. It compares nodes and parameter sizes.

Usually, during LVS verification, schematic information is used as a reference. Any errors that occur during an LVS check should be fixed in the layout to match the schematic.

The quality of LVS verification depends on the LVS rules. Good LVS rules give quality LVS verification results. An LVS check should be run at both the sub-block and block levels, to reduce complexity and to fix any problems up front.

Troubleshoot Layout Versus Schematic Errors

The first approach in troubleshooting the LVS is to verify that all of the basic components between schematic and layout match (not counting parallel components mismatched). If component matches are not resolved, then other errors such as in the net, terminal, and parameter may be the cause of this mismatch.

After components are matched, the second approach is to make sure the connections and terminals match between layout and schematic. This mismatch or match does not affect the parameter comparison, but it is more important. If connections are mismatched and the functionality is mismatched between schematic and layout, then the design has already failed.

The next important area to troubleshoot would be to verify that the parameters between the schematic and layout are matched. When the parameters match, the layout is considered a matched design.

Layout Versus Schematic Technical Details

You can use the two different tools to perform LVS checks. Specific technical details are described in the sections below.

- Diva Layout Versus Schematic
- Assura Layout Versus Schematic
- Common Layout Versus Schematic Issues

Diva Layout Versus Schematic

1. The input to the Diva LVS check is an extracted layout with all devices/sub-blocks placed and hooked up. It is recommended that the layout be DRC clean. It is not necessary to have parasitic information in the extracted netlist for the purpose of running LVS.
2. LVS can be executed using
   - The Design Framework menu: Choose the *LVS – Verify* command from the design window.
   - Batch mode in the CIW: `ivLVS`
   - Batch mode in UNIX:

     `LVS [-h] [-l` *`libname`*`] `*`cellname viewname`*
   - Remote verification in batch mode: Jobs are run on a remote server

   Options for running LVS include

   - Run directory name: The run directory (default is `LVS`) contains all the necessary files such as `si.env` (environmental control), `si.log` (simulation log file), and netlist and error files under `schematic`/ `layout` subdirectories.
   - Generating new netlist: When options are changed for subsequent runs and cellviews have not been edited, this feature can be turned off because there is no need to create a new netlist during the next run.
   - Defining the cellview: Three methods can be used to choose the cellview, point to the cellview window, point to the cellview name in the library browser, or enter the name of the cellview in the form.

- Selecting a rules file and rules library: The rules file typically resides in the technology library and the default name is `divaLVS.rul`.
- Options for rewiring: Device fixing, creating cross-references and terminals.
- Options to display output, errors: Monitor progress, backannotation, parasitic probing, and building analog or mixed extracted models.

3. Once the correct switches have been selected, click *OK* or *Apply* to run extraction. The results appear in the CIW. A long error list can be viewed by scrolling or viewed in a text editor from the `CDS.log` file. The summary (number of errors and number of warnings) appears at the bottom.
4. Errors, if any, can be located with commands under *Verify – Markers*. When *Explain* is selected, the Explain Text window opens, showing messages for LVS errors as they are pointed to. *Find* opens the Find Marker form, which has options for displaying LVS errors. *Edit – Search* shows messages for errors found by the search filter that is user-defined.

For more information, see the *Diva Reference Manual*.

Assura Layout Versus Schematic

Assura LVS is the newest Cadence design verification tool. It does not require a separate step to do layout extraction but has embedded this functionality within the LVS tool itself. It also provides for both ERC and LVS checking although some of the checking provided by Diva ERC is not supported by this tool at this time.

1. The input to LVS can be a layout with all devices placed and hooked up, including power/ground rails, guard rings, and wells. It is recommended that DRC be run and DRC violations be fixed prior to running LVS.
2. Assura LVS can be run using batch mode in UNIX.
3. To run in batch mode in UNIX:
   a. Create an RSF file (run-specific file) for the design. Usually an existing file can be copied from another job, or a template file can be edited so that it specifies the design library, cell, and view, and other run-specific details. For example, if the name of your DFII schematic cell is `mosTest` and it is located in a library called `test`, the file might look like this:

```
avLibPath = "../../testLibs"
avLibName = "test"
avCellName = "mosTest"
avViewName = "schematic"
avViewList = "auLvs schematic symbol"
avStopList = "auLvs"
```

```
avNamePrefix = “ignore”
avSimName = “auLvs”
avVldbFile = “netlist.vlr”
```

b. From a UNIX command line, type

```
dfIIToVldb assuraLVS.dsf
```

c. . For example, if you want to run LVS on a DFII layout and schematic cell called `mosTest` located in a library called `test`, the file might look like this:

```
;DFII Data
?inputLayout (“DFII” “test”)
?cellName “mosTest”
?viewName “layout”
?rulesFile “assuraLVS.rul”
?runName “mosTest”
?workingDirectory “assuraRunDir”
```

d. From the UNIX command line, type

```
assura assuraLVS.rsf
```

A directory called `assuraRunDir` will be created by Assura and will contain the Assura files from this run.

e. If you want to run LVS on a GDS2 stream file called `mosTest.gds`, the assuraLVS.rsf file might look like this:

```
;GDS2 Data
?inputLayout (“GDS2” “./mosTest.gds”) ;specify full path to gds file
?viewName “layout”
?rulesFile “assuraLVS.rul”
?runName “mosTest”
?workingDirectory “assuraRunDir”
?avrpt t
```

f. Once the batch job is complete, access the results through the *Assura – Open Run* command. Enter the path to the run results, and *OK*. The LVS Error Report window appears along with the Error Layers Window (ELW). Keep the ELW open.

4. Information for running an interactive Assura LVS is given below.

5. The form that appears when you choose *Assura – Setup – LVS* does not need to be completed, and allows a mechanism for users to customize settings which ultimately end up in the RSF file. A user may use this form to customize the RSF, or instead, use an include mechanism from the Run LVS form to include the same and/or different RSF options. Which options and settings to be used depends on the rules that come from the foundry.

6. The *Assura – Run LVS* command, brings up a form with several fields and options.

Note: At the top of the form are the *Load State* and *Save State* commands. One may load a state file, which seeds the entire form with saved settings. Once completed by a user, the form may be saved to allow easy retrieval of the same

settings.

a. By default, the form gets the currently open schematic and layout view names. You may change that if required.

b. The default run directory is `.`, which is the working directory where the software was started. A directory name can be added under which all Assura files will be created (for example, `. ./assura/LVS`) if desired.

Note: This directory must be created manually from UNIX. The *run* command will not create this directory.

c. Run name: The default is the same as the cell name, which can be overridden if desired.

d. RSF name: This is the output filename that the form, once executed, will write to. The default is `cellname.rsf`.

e. Specify the LVS rules (Extract rules and Compare rules): There are two methods available to select the rules. One is to use *Assura – Technology* to create a technology library, which is selected in the Run LVS form. The other more common method is to not use the *Technology* method by leaving the *Override Technology Selection* choice selected, which allows the user to specify a specific path in the *Rules File* section.

f. Set switch names: After the rule files is specified, click on *Set Switches* for a list of choices that the rules define (if any). Whether to select switches or not depends on instructions for using rules that are provided as part of the rules obtained from your foundry. Typically, there is an *?extract* switch to select or deselect extraction.

g. RSF include: If desired, one may include a `user.rsf` file, which will be inserted at the top of the complete RSF file that the UI creates and submits to the Assura tool. A foundry-supplied RSF could be specified, so long as every line contains legal input. The `runName.rsf` file is assembled by first including your `user.rsf` file, followed by the RSF from the form settings. However, if any duplicate options occur in the `user.rsf` file compared to what the form will generate, the form values will override the entries in the included RSF. After submitting the DRC form, examine the `runName.rsf` file to understand exactly what `user.rsf` options will be overridden.

h. Template file: A template file is generated by using the form that appears when you choose *Assura – Setup – LVS*, which can then be included via this field on the Run LVS form.

i. The *View list* should typically include the auLVS, schematic and symbol views and the *stop list* should be auLVS. You should also select *Ignore namePrefix from CDF* and *avSimName* should be set to `auLVS`.

7. Run LVS: Once the correct switches, rule deck, and other settings have been selected, click *OK* or *Apply*. A Progress Form will appear. You may want to click on *Watch Log File* to observe progress of the LVS job. A window will pop up when the job completes, stating that the job has finished. The pop-up window allows you to view the results at that time. A choice of *Yes* will bring up the Assura LVS Error Report (a summary of all the errors/ mismatches in the design) and the Error Layer Window.

8. If the Error Layer Window is not already open, use the *Assura – Open ELW* command. Leave this form open until completely finished with the Assura LVS results.

9. The left side of the ELW lists which errors are violated and how many for each rule. Click on any error. Click on the right or left arrows to zoom in to each error. Also observe that one or more cell names appear in the right side of the ELW, along with an error count next to each cell. These are the actual cells that contain the errors. Choose a cell name. Notice the *Open Report* command is then selectable. Selecting this will cause a text report of those errors to appear, which includes hyperlinks to specific coordinates for each error. These hyperlinks or the left-right arrows can be used to zoom in to each error.

10. Assura LVS does not understand pins that are not labeled. Therefore you need to make sure that *Pin Label Shape* is set to *Label* in the Gen from Source form at the time of layout generation.

11. Assura rules are written in Assura syntax. If the user desires to submit Diva rules to the Assura tool, in many cases it will work without any changes. The Assura tool can translate most Diva syntax into Assura syntax. Sometimes there are differences, and the CIW output often indicates which rules are problems and often what to do about them. Because of differences in Diva and Assura capabilities, The message might say, for example, to write a Diva rule differently, or to override and ignore the difference.

    Note: To use Diva rules consistently it is suggested that they be converted to Assura syntax by setting *?compileOnly t* in the RSF file. That will cause the Assura tool to translate the Diva rules and write them out to a file in Assura syntax. This file can then be examined, modified, and enhanced to take full advantage of all the features that the Assura tool offers.

Refer to Assura documentation, and if possible, attend an *Assura Verification* training course. While similar to Diva, Assura verification still has differences in UI and in the methods used to debug with a true hierarchical verification tool.

Common Layout Versus Schematic Issues

LVS involves netlisting the schematic. Hence, it is essential to make sure that there are no netlisting issues (for example, due to viewname not being in the viewlist).

Exclusive-OR (Optional)

Exclusive-OR verification is usually performed when a design is nearly complete or when a re-spin design needs only a few changes to the layout. Exclusive-OR verification is used to verify that the modified area of design is what was expected. The differences are highlighted when compared to the layout data before modification, which the designer can then study to see the exact changes that were made. This helps ensure that the designer did not accidentally make a layer change in an unintended area.

Parasitic Extraction

Parasitic Extraction and Backannotation

The purpose of this task is:

- To extract device components with resistance and capacitance parasitics in order to prepare the layout netlist for post-layout simulation
- To backannotate the schematic with parasitic information for the purpose of simulation or documentation.

Note: Parasitic Extraction requires an optional product called ‘Diva RCX’. The product Assura RCX can also be used, but is not covered in this appendix.

At this stage of the design, the LVS comparison between schematic and layout must be clean; or else the output netlist may give unexpected results when running postlayout simulation. The extraction in this stage is intended to generate a netlist with true layout characteristics. This means device components, connections, and parasitic components must be extracted. The parasitic information is combined with the original logic circuit to produce a more accurate simulation of the circuit. As processes shrink and the feature size goes down, the parasitic device effects become more and more critical to the correct functioning of the circuit. The design environment should facilitate the collection of the parasitic devices to allow for an efficient method of combining these with the original circuit in the simulation environment.

The device components and connections are the intentional components that are already modeled in the schematic. The unintentional components that may exist in the layout design can be resistance, capacitance, diode, and BJT. These unintentional components can be considered *noise components* because they will generally generate unwanted characteristics. Post-layout simulation is needed because parasitic components can change the design performance. This simulation is sometimes referred to as design characterization.

The accuracy of the data depends on the extraction rules. The designer may need to check design rules and make sure that the equations in the rules are correctly written. Inaccurate equations may lead to inaccurate component values. Also verify that each layer the rule uses is the intended one. A parasitic netlist extraction should be run at both the sub-block and block levels, to reduce complexity and to fix any problems as early as possible.

Parasitic extraction may need to be run several times on different portions of the circuit. For example, digital designs may only be interested in characterizing the lumped parameter delay effect while an analog simulation may be concerned with other information. Some specialized extraction may be necessary when studying transmission line effects of critical signals. Generally the analysis is performed by the original design engineer but the layout designer

may be asked to perform the extraction tasks. In addition, layout changes may be required based on this analysis.

The coefficients and capacitance overlaps are defined by the foundry and represented in the extraction rules file. There are two options available which are mutually exclusive - one for capacitance (C) only and one for a combination for resistance and capacitance (RC). When the user turns on the extraction tool one or the other will need to be selected. The default is no parasitic extraction, which means that only designed devices are extracted for LVS comparison. In the usage of the tool Diva refers to these options as *switches*. If you have selected the C or RC switch and wish to return to the non-parasitic extraction option, you may have to erase the option in the switches field if there is no ‘default’ option in the switch list.

Once extraction has completed an extracted view of the layout is generated. The extracted view contains the parasitic devices as part of the connectivity data. The parasitic devices can be viewed as symbols in the layout view. These symbols are typically copies of the schematic symbol, which are scaled to match the layout size, and are called an ivPcell view. The extracted view is equivalent to the schematic view in that all discreet devices and nets are in the extracted view. Because LVS has been completed prior to Parasitic Extraction, the extracted view and schematic view are equivalent. By adding the extracted parasitic devices to the extracted view we have a complete netlist ready for re-simulation.

Implementation

The device extraction technology is contained in Diva-LVS. This tool will be used to update the schematic for documentation purposes. The interconnect capacitance extraction technology is in Diva-RCX. This tool will be used to provide simulation information.

1. From the Virtuoso XL layout view choose *Verify – Extract*. Choose the *C* or *RC* switch. If you want to retain the LVS extracted view then change the name of the extracted view in the form to give it unique name such as extracted.c or extracted.rc. From there the Diva Extraction tool will then extract the discreet and parasitic devices and create an extracted view.

2. If there are errors or warnings, they will be noted in the CIW window. To examine the errors bring up Marker Form from the *Verify – Markers – Find* menu. Choose each error or warning and view the message. Some errors that might be encountered include missing wells or multiple physical terminals with the same name. Once all errors are cleaned up and the warning messages are understood, then the extracted view is ready for re-simulation.

3. Re-simulation involves two options – backannotation and re-simulation. Backannotation simply annotates the schematic with text to show the total capacitance for the net. This text does not allow a simulation to function with the text information, but can be used for a visual check by the user. Re-simulation actually uses the extracted view as the *source*

schematic information. The extracted view has most of the information from the schematic view along with the parasitic devices from layout.

4. With a Virtuoso Schematic Composer window open you can choose the Backannotate button from the Diva LVS form. The schematic view of the cell will be updated with text showing the total capacitance for that net. This is calculated from each of the capacitance elements present in the extracted view for each net. If the *Backannotate* button is not present on the LVS form then you need to restart DFII, making sure that the environment variable 'CDS_Netlisting_Mode' is set to *Analog* or *Compatibility*. In this mode the Analog Design Environment interface options are present.

5. Re-Simulation involves access to the Analog Design Environment. This step will be discussed in detail in the Analog/Mixed-Signal (AMS) flow as it relates to the initial simulations performed on the circuit. See Diva Flow: Working through the Extended Design Example chapter in the *Cadence Parasitic Simulation User Guide* for more details.

For more information on this topic, check the *Diva Parasitic Simulation* manual or the *Diva Reference*.

Stream Out the Data

Once the layout is complete, it is often a requirement to translate the layout data to a standard format either for the purpose of mask generation or for use in another design environment using other tools outside of DFII (Cadence Design Framework II database). Typically the output needs to be in GDS format which is the industry standard. The Polygon input/output database translator (PIPO) is used to perform the stream translation.

Sometimes it may also be required to translate or import a design that is in GDS format into the Design Framework II for the purpose of block integration, across different design environments. The PIPO translator can also be used to import or stream in GDS data.

Streaming Out using Polygon Input/Output Database Translator

1. Bring up the Stream-Out form from the CIW (Choose *File – Export – Stream* from the CIW pull-down menu).
2. You should provide the following information:
   a. Run directory where the streaming information will be generated.
   b. Library name, Cell name and View name
   c. Output (gds) file name.
   d. Log file name.
3. Click on the button *User-Defined Data* at the top of the form and provide the following information:
   a. If you want to retain the pins, choose either *geometry* or *text*, else choose *drop* in the field *Convert Pin to*.
   b. If you chose *text* option for the pin conversion, you need to provide a Pin Text Map Table (**optional**).
   c. You should provide a non-zero number in the *Keep pin information as attribute number* field. The same number should be used for streaming in to get the pin information back.
   d. Provide the file name for cell name mapping in the field *Cell Name Map Table* if you want to change the cell names or map them in a certain way (**optional**).
   e. Provide the name of the file containing the layer mapping information in the field *Layer Map Table.*

f. Provide the file name for mapping text fonts if you need specific fonts (optional).

g. Click *OK* in the User-Defined Data form.

4. Click on the *Options* button to bring up the Options form

a. Choose *merge polygon* under *Convert Pcells to Geometry* to generate a flat layout (choose *No Conversion* if you want preserve the instances as static cells).

b. Choose *preserve* under *Case Sensitivity* to maintain the same case in the streamed data.

c. You can choose to retain the reference libraries for the design by selecting the *Retain Reference Library* option (make sure the reference libraries are defined in `cds.lib`).

d. Leave everything else to the default values and click *OK* in the Options form. You can choose the *keep pcell* option if you want to restore the pcells when you stream the data back in to DFII (assuming you do not make any changes in the gds file). In that case, you need to choose *No Conversion* in the *Convert Pcells to Geometry* field, turn on the *keep pcell* option and provide the path to the `ROD` directory (directory for storing the ROD information to be used when the data is streamed back in) in the Options form. Make sure you delete any existing `KPDIR` directory that stores the pcell/ symbolic contact information.

Note: There are some problems with the *keep pcell* option right now, notably the property bags do not get copied over into the streamed library when you stream the data back in, and also abutment properties for the pcells are not retained when the cells are streamed in.

5. At this point you may choose to save the settings entered so far, into a stream-out template file. This option is available at the top of the main Stream Out form (*Template file Save*). The advantage of saving this information in the template file is that it can be reloaded at any point (*Template file Load*) and necessary changes may be made without having to re-enter all the data.

6. Click *OK* in the Stream-Out form. Once you get the message regarding completion of the stream out process, look in the PIPO log file to check for errors/ warnings if any.

Note: Options to preserve the DFII specific data are optional (for example, text font mapping, cell name mapping, pin text mapping etc.). Specifically, you should choose *geometry* option for pin conversion. Also, simply creating a gds file for generating masks does not require most of these options. However, if you are streaming the data back in, you need to make sure the necessary CDBA information is preserved.

Streaming In

1. Bring up the Stream-In form from the CIW (Choose *File – Import – Stream* from the CIW pull-down menu).
2. Provide the following information:
   a. Run directory where the streaming information will be generated.
   b. Input (gds) file name and top cell name.
   c. Output library name: Make sure the library exists and is attached to the appropriate technology file. The library should also be defined in `cds.lib`.
   d. ASCII technology file name for the library.
   e. Log file name.
3. Click on the button *User-Defined Data* at the top of the form and provide the following information:
   a. Provide the file name for cell name mapping in the field *Cell Name Map Table* if you want to change the cell names or map them in a certain way.*
   b. Provide the name of the file containing the layer mapping information in the field *Layer Map Table*.
   c. Provide the file name for mapping text fonts if you need specific fonts.
   d. If you want to restore the pin information, provide the same attribute number that was used for streaming the data out, in the field *Restore Pin Attribute*.
   e. Click *OK* in the User-Defined Data form.
4. Click on the *Options* button to bring up the Options form
   a. Choose *preserve* under *Case Sensitivity* to maintain the same case in the streamed data.
   b. Turn on the *keep pcell* option if you want to restore the pcells and if the data was streamed out with this option, assuming no changes were made to the gds file.
   c. You can choose to retain the reference libraries in the streamed in data by selecting the *Retain Reference Library* option if the data was streamed out as well with this option. Make sure the reference libraries are defined in the `cds.lib`.
   d. Leave everything else to default and click *OK* in the Options form.

5. At this point you may choose to save the settings entered so far, into a stream-in template file. This option is available at the top of the main Stream In form (*Template file Save*). The advantage of saving this information in the template file is that it can be reloaded at any point (*Template file Load*) and necessary changes may be made without having to re-enter all the data
6. Click *OK* in the Stream-In form. Once you get the message regarding completion of the stream in process, look in the PIPO log file to check for errors/ warnings if any. Make sure the warning messages make sense and are acceptable.
7. You can now bring up the imported design in Virtuoso XL from inside DFII to make sure the data has been translated correctly and has all the required layers and shapes.

For more information on data translation, consult the *Translator Reference Guide*, Stream chapter.

Design Modification

Engineering Change Order (ECO)

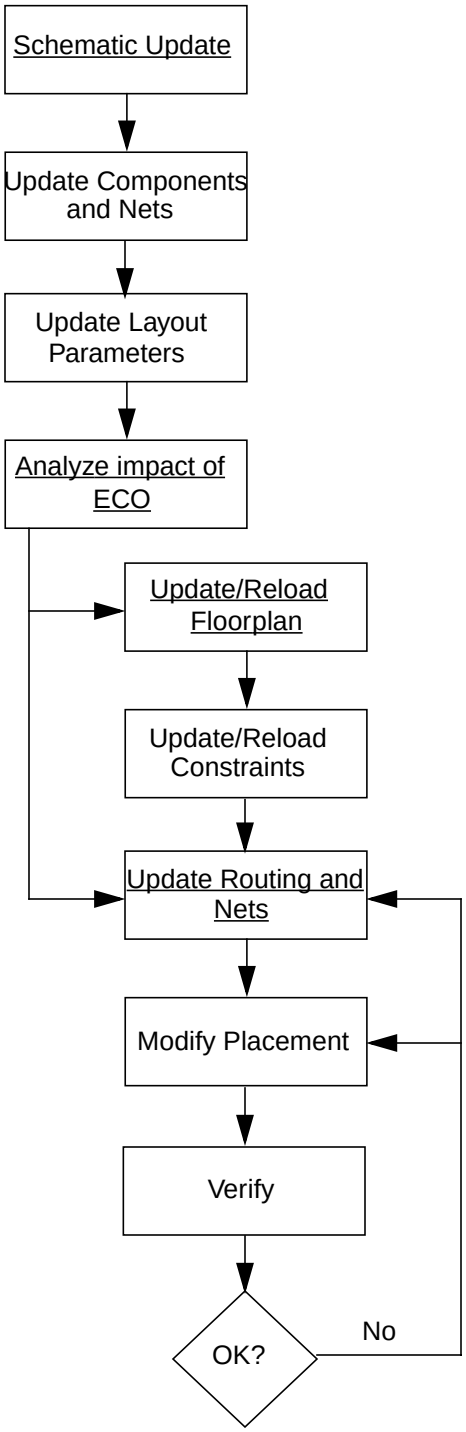
There is often a need to iterate and make changes to the design at various stages of the design process. These may be changes which reflect design decisions based on layout issues and can be implemented at the constraint or layout level. Other times performance analysis will uncover problems that might involve significant logic changes which in turn could impact the placement and/or connectivity of the design and require an ECO. The ECO will start in the Schematic with a logic change such as adding components or nets, or replacing an existing component. The change may also simply modify the value of one of the component parameters, requiring a modification in the size or shape of the physical implementation. It is easier, quicker and more convenient to implement ECOs in an integrated design environment.

There are several approaches to completing an ECO. Typically the goal is to redo as little of the already completed work as possible. If there is a logic change which significantly impacts the layout placement then a complete redo of the placement and routing steps may be required. On the other hand, if the changes are due to certain layout constraints not being met or due to performance and reliability issues requiring changes in device parameter and routing of certain nets, then the changes can be made incrementally without disturbing most the floorplan, placement and routing. Given below are some general ECO scenarios commonly encountered in a design flow.

- Functional verification and/or postlayout simulation requires logic change (additional nets and components, removal of some of the old nets and components).
- Postlayout simulation shows that some device parameters need to change in order to meet performance targets or to improve signal integrity.
- Postlayout simulation shows that certain nets need to be rerouted or shielded or their width, spacing and other attributes need to change for performance, reliability and signal integrity reasons.
- Manual inspection/constraint status analysis shows that some constraints were not met while using an automated approach.
- Later design enhancements need to be rolled into an existing layout.

For detailed information on performing an ECO See “Engineering Change Order Cycle Detailed Instructions” on page 244.

The following flow diagram illustrates the steps for the ECO process (click on a link to view more detailed information):



ECO Case Study

After going through the first pass layout design flow, and spending significant time hand tuning placements and routing, the schematic for the block being worked on has undergone some changes. The block had been laid out initially in Virtuoso XL for placing certain critical components, and then sent to the automatic placer and router.

Placement: The ECO schematic had a couple of additional gates totalling less than 10 devices. The layout parameters, mostly width, have also been changed for some of the devices in order to meet performance targets.

1. The first thing to do is run the *Update Components and Nets* command in Virtuoso XL in order to generate the additional devices, and to extract the corresponding connectivity.
2. Also run *Update Layout Parameters* to make sure the parameter changes in the schematic get updated in the layout devices.
3. Run a connectivity check against the source (i.e., the schematic) to make sure that the schematic and the layout are synchronized.
4. Now, to get an optimal placement for the unplaced devices, start the placer after locking down the critical devices.
5. With all other settings unchanged, choose *ECO mode* in the placer. The placer generates a placement with minimal change in the existing placement, while the locked devices are not touched at all.
6. Investigate the placement to make sure that the additional devices have indeed been placed optimally, and hand tweak those that are not. In particular, look for cases where the logical hierarchy has been disturbed. It was a good idea to have generated rows in the Virtuoso custom placer having less than 100% utilization in the initial layout generation phase.

Routing: The way the placer works in the ECO (and regular) mode with routed design is as if the routing doesn’t exist. So running it through the Virtuoso custom placer ECO mode is the same as merging or reading in a modified placed view (design file) into an existing the Virtuoso custom router wires file. You should pay attention as the routes for the new devices are added.

This was only a bit tougher than the case a week earlier where there was no schematic change, only a routing change (optimizing critical nets. shielding or increasing width/ separation of certain nets etc. based on simulation results). In that situation, the placement changes were done inside the Virtuoso custom router environment (for example, doing a push and shove of some devices to make room for a net, and some device swapping to shorten a net).

Managing the Design Environment

Design Naming Conventions

One of the issues in making a library that will work with all of the tools is to adopt some key naming conventions.

- Layer names should never begin with a number
- All layer names should contain only alphanumeric characters
- All parameter names should contain only alphanumeric characters
- All device names should contain only alphanumeric characters and underscore characters
- All parameter/device names should be no longer than 15 characters
- The instance prefix for a given device should be an alphabetic character

Pcell Naming Conventions

The parameterized cells in the sample pcell library are implemented with SKILL functions which permit a high degree of functionality and flexibility. The SKILL functions are conceptually no different from any other SKILL function and are thereby subject to conflicts both in the names of the functions and in the names of variables used in the functions. If you are planning to build your own pcells you should be careful to ensure that there are no conflicting Cadence names. The recommended method of avoiding potential conflicts is to use naming conventions for the site.

Cadence employs a naming convention for all SKILL functions. Each function is named with a two or three letter prefix followed immediately with the function title which is started with a capital letter. In addition some functions begin with an _ (underscore) character. Cadence recommends that customers always use a prefix of a few characters and that this prefix begin with an uppercase character. This will ensure that there will not be any conflicts with Cadence names. A similar prefix should be used for any global variables. The prefix should be unique to a particular kit and used for all function names and variables in that kit. For example: a kit is needed for the TSMC 0.25-micron RF process. A prefix might be T25R and the SKILL file might be called T25Rmos.il. This prefix is short, easy to type and supplies some information about the kit itself. To carry out this convention some of the functions might be called T25RcreateMosDevice() and T25RsortValues(). Note the use of upper case letters to avoid conflicts with Cadence names and to provide an easy to read name that is descriptive of the function.

For more information on designing pcells refer to the Parameterized Cell Reference manual.

Design Framework II Library Objects

A good library naming convention for customers might be to start the name with an upper case letter. Cell names differ among Cadence libraries. There should be no problem with overlapping Cadence names so long as the libraries themselves have unique names.

Backup and Archive

You should back up your design frequently so that you will be able to recover the design in the case of a computer failure or need to revert to a previous version of the design in process. One method is to provide a revision control system. You would backup the design by checking in versions on a periodic basis. A secondary backup system may still be desirable that does not depend on user action.

The following list of files and directories that are contained in your design hierarchy should be backed up and maintained using a revision control system:

- The working directory and all of its files.
- The design library and all of its files.
- The `cds.lib` file
- Any do files that you wish to save.
- The display.drf file if it has been modified.

Once the design is completed an archive can be created so that, if the need arises, the design and design environment can be re-established. Archive differs from backup in that you may need to preserve some of the derived files such as GDSII out. This will help ensure that the archived data will be usable in the future even if the tools and technologies change. It is also important to attempt to capture information about the intellectual property that was invented or discovered during the design and layout. This information can prove useful for new designs that run into similar problems and performance issues.

Data Preparation Detailed Instructions

Schematic Setup Recommendations

For efficient data and design management the schematics and the corresponding layout views should be in the same library.

The pcells used in the schematic (and the layout) should point to the technology library rather than to any local copies to prevent any discrepancy. The library should be version controlled and all callbacks should be re-triggered each time a change is made in the library.

The schematics should roughly follow the desired floorplan in order to take full advantage of the connectivity driven layout tools.

Preparing the Schematic for Chaining and Folding

The following approach should be taken to allow folding/interdigitation of devices:

1. The number of fingers for the devices to be folded or interdigitated should be set to one (1) because a fingered device can not be broken into separate instances.
2. Either the width per finger or the mfactor should be adjusted to make sure the total width remains the same.
3. To maintain a given width per finger, either the folding threshold should be set to the value or the mfactor should be multiplied by the number of fingers.
4. Modifying the mfactor to adjust the number of fingers will ensure that the width per finger remains the same. This will give you the flexibility to chain/interdigitate the fingers or arrange them in symmetrical/co-centric fashion.

Device Preparation

1. To allow flexibility in laying out resistors, capacitors, and inductors, the sfactor (s, S) property may be added to these components in the schematic, instead of generating them as one single instance with series/parallel segments.
2. Spare devices in the schematic should be created as per the methodology described in the placement section in “Device-Level Floorplan and Placement” on page 125.

Design Environment Setup

1. Create new directory for the design and `cd` into this directory. If you have a common design area then just cd to that directory.
2. Copy the sample `.cdsenv` file into the working directory.The `.cdsenv` file can be customized by creating or modifying an existing `.cdsenv` file. Environmental variables can be set through the Display Options, Editor Options and XL Options form from the Virtuoso XL pull down menu. The options can be saved to a `.cdsenv` file. A sample `.cdsenv` file with is located at <install_dir>/tools/dfII/samples/.cdsenv. The `.cdsenv` file should be copied to the design directory before beginning the design.
3. Copy the sample `.cdsinit` file to the working directory. The `.cdsinit` file is used to define and/or load customized SKILL code. When DFII is started, the software searches for the `.cdsinit` file in the current directory or the home directory unless a local file has been setup during installation. A sample `.cdsinit` file has been provided in <install_dir>/tools/dfII/samples/local/cdsinit.
4. Copy the sample `cds.lib` to the working directory. The `cds.lib` file contains the library pointers that are displayed in the Library Manager browser. A `cds.lib` file will need to be created for the libraries that will be used in the design. The `cds.lib` file can be built or edited either manually or through the Library Manager. Refer to the "Library Setup" for more details
5. A display.drf file is provided in the install hierarchy. Copy the `display.drf` file to the local design directory. The display.drf (display resource file) specifies how layers appear on the display. To customize the display choose *Edit – Display-Resource Editor* from the LSW.You can edit the style, colors, and stipple patterns of the Valid layers defined in the technology file. Layers that are not defined as Valid in the technology file can be edited through *Tools – Display Resource-Manager* from the CIW menu.

ROD Sample Library Pcell Setup

1. Use the sample pcell library shipped with the software. The library is located at <*install_dir/tools/dfII/samples/ROD/rodPcells/components/device*>.
2. Install the `Sample Parameterized Cell` Library.
3. Set the layers and rules in the technology file through the installation form.

Library Setup

A design environment consists of a minimum of two libraries. One library is for the design and the second library is the reference library, containing symbol and layout views for devices

(pcells) to be used in the design. The other views such as those required for netlisting and simulation of the design are also available in the reference library. A directory with the name of your design library, for example design will need to be created.

1. In the CIW choose *Tools – Library Path Editor* to create the `cds.lib`. Or from the xterm window from where the software was started you can type `cdsLibEditor` to bring up the Library Path Editor. One line should include the `cds.lib` file for the reference library and a second line should point to the library name chosen in step one for the design library. The file would look something like:

   ```
   INCLUDE <pathname_to_reference_libary>/cds.lib
   DEFINE design ./design
   ```

2. **T**o attach the technology library to the design library choose *Tools – Technology File Manager*. In the Technology File Tool box choose *Attach*. In the Attach Technology Library to Design Library form specify the *Design Library name* and the *Technology Library name*.

   Note: Another way to attach a library is when you create the design library choose the Attach to an existing technology file option.

3. Check that the UNIX permissions are set properly for the design library. Certain users may not need write access to the schematic or symbols but may need write access to the design directory to create the layout. The UNIX command `ls -l` can be used to view access rights.

Other Setup files

1. The translation rule file is used for translating from DFII to the Virtuoso custom router. For more information about translation rules, refer to "Translate Design Detailed Instructions" on page 215.

2. You will have to create the rules file for running Diva. These rule files have a .rul extension and may include any of the following rules:

   a. DRC rule file

   b. Extraction rule file

   c. LVS rule file

   d. Density rules

   e. Antenna rules

   Note: Check with the foundry or foundry web site and download the latest rules for the design library. Each company may have additional rules decks that go beyond the

minimal requirements of the foundry decks.

3. Mapping layer files are used for importing from or exporting to GDSII format. The GDSII formatted stream file will usually be kept in its own directory within the design area.
4. Simulation setup files are left over from the schematic creation task or part of schematic design verification.

Create Footprint Detailed Instructions

1. Open the schematic in the Virtuoso Schematic Composer, and make sure the schematic view has been extracted for the latest changes.
2. In the schematic window choose *Tools – Design Synthesis – Layout XL* to start Virtuoso XL.
3. In the Start Up option form choose to create a new layout cell view, unless there is an existing cell view to be used.
4. In the Create New form specify the Library, Cell, and View name. By default the cell name is the same as the schematic and the view name is layout. If you selected to open an existing cellview then the Open File form will appear. After selecting *OK* the layout window appears
5. To create a new template or to load an existing template in the layout window choose *Design – Gen from Source*.

   The Layout Generation Options form allows you to create I/O pins, the bounding box and the devices/instances automatically from the schematic. The instances maintain their connectivity in the layout as per the schematic. This capability is the crux of connectivity driven design.

   In the Layout Generation Options form you can specify the components that will be generated in the layout. You can specify that the bounding box and the I/O pins are generated but not the instances. It is recommended you generate the devices as well in order to get a better estimate of the boundary/block size.

   Note: One way of incrementally generating the layout (i.e., pins and boundary followed by instances) is to use the *Connectivity – Update Components and Nets* command. The Layout Generation Options form appears after invoking the *Update Components and Nets* command.From the Layout Generation Options form you can set any of the options. This ensures that any pins or instances generated earlier will not be deleted (regeneration using the regular *Design – Gen from Source* command deletes all existing pins and instances).
6. Specify the bounding box layer and purpose (typically prBoundary drawing) as well as the *Boundary Height* and the *Boundary Width*. Optionally, either the *Boundary Width* or *Boundary Height* and the *Aspect Ratio* can be provided as well. Another option is to provide the *Utilization percentage* along with *Boundary Width* or *Boundary Height* or *Aspect Ratio*.

   The bounding box size can be based on a rough estimate and adjusted later. It is recommended that the bounding box be adjusted/updated after generating the devices and doing a rough, first pass device placement.

Note: Virtuoso XL automatically estimates the boundary as a summation of the bounding boxes of all the instances and applies the aspect ratio to it. However, it does not account for folding, chaining and abutment.

7. Choose the *Pin Type* for he I/O pins. The I/O Pins can be either *Geometric* or *Symbolic* (provided symbolic pins have been defined in the technology file).

8. For each I/O pin specify the layer purpose pair, height, and width. By default, the height and width displayed are the minimum width set for the layer in the technology file. Only width can be specified for symbolic pins because they are always square. You can propagate a single value for the Pin Type, Layer/Master, Width, Height, and number of pins generated and then modify each pin as needed.

   Note: All routing layers including the boundary layer must be defined in the technology file under lxExtractLayers. If the layers are not in the lxExtractLayers, then they will not be extracted and connectivity will not be preserved. You must also add the 'abutment layer' (usually diffusion) to the lxExtractLayers, in order for devices to abut (share their diffusion).

9. The Layout Generation Options form only displays the pins that are in the schematic. If additional pins are required for layout purpose only (for example, spares and feedthroughs), they can be added using the *Add Pins* option. Net name, layer and size information are needed for these additional pins.

10. Input pins, including power pins, may also be used as feedthroughs, especially for arrayed cells or blocks. Create two (or more) instances of the same pin by changing the *Num* text field.

11. Use the Virtuoso custom placer *Place – Pin Placement* command to place the pins along the prBoundary with the exact location and spacing. This is a step that can also be postponed until the actual placement step.

12. Use the *Transistor Folding* and *Transistor Chaining* option to fold and/or chain devices automatically.

    Note: To fold or chain devices either automatically, you must define the Component Type for each MOS device. Component Types are defined from the *Design – Component Types* command. Each device is required to specify the component class or type, parameter name for width (lxWidth), the maximum allowable width (lxMaxWidth), and the layer purpose pair for the active/diffusion layer in the device (lxActiveLayer). All other fields are optional (see "Automated Placement using the Virtuoso Custom Placer" on page 208 for more details).

13. Transistor folding allows you to split a device that has a large width. Folding is driven by layout requirement (for example, to align devices in a row by making sure their widths do not exceed a certain threshold). Folding generates separate instances that are abutted

by default but can be placed apart if required. Multiple fingered devices in the schematic or in the layout cannot be folded.

14. Transistor folding computes the number and width of fingers based on the maximum device width specified in the Component Types form. Interactive folding from the *Edit – Transistor Folding* command allows the user to specify the number and width of the fingers for a device. For folding, it is necessary to set the environment variable mfactorSplit to true.

Note: Because folding is not reflected in the schematic, creating fingers in the schematic might be a preferred for simulation purposes of sensitive circuits. For the same reason, instances of folded device should be as close as possible to ensure accurate parasitic backannotation.

15. Transistor chaining abuts devices that can share diffusion. This can include folded devices. Automatic chaining verifies that the complementary NMOS and PMOS transistors are vertically aligned (to minimize routing) and then maximizes the number of abutments. Devices must be set up for abutment.typically exists in the pcell code. Fingered devices can not be broken into separate instances.

16. To control which devices should or should not abut, then it is recommended that *Transistor Chaining* and *Transistor Folding* be turned off.

17. If there are spare devices in the schematic, it is recommended that they not be generated with the rest of the devices if folding and chaining are turned on. This would cause the spare devices to be abutted and chained with the active devices. There are several ways to handle spare devices., two of which are described below.

a. Identify the spare devices in the schematic that has write privileges.

b. Choose *Edit – Properties – Objects*. In the *User Property* section choose *Add*. In the Add Property form change the *Name* to *lvsIgnore*, the *Type* to *Boolean*, and the *Value* button should be turned on.

c. Choose *Design – Gen from Source*, and turn *Transistor Folding* and *Transistor Chaining* on. The spare devices are ignored.

a. It is preferable to generate spare devices up front in order to get a better estimate of the layout area and placement.

b. To generate the spare devices the corresponding `lvsIgnore` property must be turned off (there is no need to delete the property).In the schematic choose *Edit – Property – Objects*. In the Edit Object Properties form turn off the *Local Value* of the `lvsIgnore` property.

c. Choose *Connectivity – Update – Components* and Nets**.** In the Layout Generation Options form turn on *Instances* and turn off *I/O Pins* and *Boundary*. The spare devices are generated in the layout.

You can choose to fold/chain the individual spare devices which may help ensure optimal placement, but make sure they are not abutted with the placed device chains.

If the schematic is write protected and there is still a need to use chaining and folding options then follow the steps below;

a. Choose *Design – Gen from Source* with *Transistor Folding* and *Transistor Chaining* turned on. The spare devices will be included with the placed devices.

b. Choose the spare devices in the schematic and delete them from the layout. You may need to manually fix the chains broken due to the removal of the spare devices.

c. Choose *Connectivity – Update – Components and Nets*. In the Layout Generation Options form turn on *Instances* and turn off *I/O Pins* and *Boundary*. The spare devices are generated in the layout.

18. If folding and chaining options are not selected, the devices are generated and placed based roughly on the schematic topology. However, if the devices have been generated with automatic folding and chaining option, exact schematic topology may not be preserved.

Note: *Edit – Place from Schematic* will not work with folding and chaining.

19. Once the components have been generated the boundary may be adjusted based on a rough placement while accounting for interconnect. To stretch the prBoundary interactively choose *Edit – Stretch*. Make sure that the prBoundary layer is selectable in the LSW. Choose the edge to be stretched and drag the cursor to the new location.

20. Choose *Connectivity – Check Against Source* to check the layout against the schematic. The following should be verified:

- ❑ All schematic devices and components have been converted to layout.
- ❑ All parameters match between the layout and the schematic.

Note: If the layout was generated with the *Instances* option turned off in the Layout Generation Options form, then the *Connectivity – Check Against Source* command will not check component mismatch between the schematic and the layout. Refer to the post placement steps in the ‘“Pin Placement” on page 199’ section for more details.

Note: Certain parameters may not match between the layout and the schematic, (for example, certain stretching parameters may only apply to the layout and not to the schematic). To suppress error messages pertaining to parameters is to add the property

‘`lxIgnoredParams` property to these parameters, on the schematic. This can be done either by editing the base CDF or the *Add property* option in the property editor form for the master.

21. You can re-use and regenerate your layout based on the pin and prBoundary information. Once you choose *OK* all pin and prBoundary information from the Layout Generation Options form are saved in the cellview and then can be dumped to an ASCII template file To create the ASCII template file choose *Design – Save Design To Template*. In the Template File form turn on *Boundary* and *I/O Pins Location*s. Specify a template file name `template.lxt`.

Power, Guard Rings, and Wells Detailed Instructions

Creating a Single Power Rail

1. Identify the area where PMOS transistors are to be placed (over an n-well). Choose *Create – Rectangle* to draw a rectangular power rail. In the LSW (Layer Selection Window) choose the drawing layer (for example, *metal1* drawing).

   Note: You can also create the power rail using the *Create – Polygon* command.

2. To create the power pin choose *Create – Pin*. Specify a rectangular pin and *Terminal Name* (for example, vdd or vcc). If you created the power rail on *metal1* drawing, then you would want to create the power pin on *metal1* pin. Create the pin co-incidentally on the rectangle that you created using the *Create – Rectangle* command, and the periphery of the block. The width should meet the top-level requirements or be based on calculation that shows that the voltage drop is acceptable.

   Note: If you created a polygon in the first step then change the mode to polygon in the Create Shape Pin form.

Creating a Single Ground Rail

1. Identify the area where NMOS transistors are to be placed (over p-well or substrate). Choose *Create – Rectangle* to draw a rectangular ground rail. In the LSW (Layer Selection Window) choose the drawing layer (for example, *metal1* drawing).

2. To create the ground pin choose *Create – Pin*. Specify a rectangular pin and *Terminal Name* (for example, vss or gnd). If you created the ground rail on *metal1* drawing, then you would want to create the ground pin on *metal1* pin. Create the pin co-incidentally on the rectangle that you created using the *Create – Rectangle* command, and the periphery of the block. The width should meet the top-level requirements or be based on calculation that shows that the voltage drop is acceptable.

Creating Multiple Power and Ground Rails

1. There may be more than one power or ground rail based on the transistor arrangement or floorplan (for example, for the desired row pattern in P-N-N-P, there would be two power rails for the top and the bottom P rows, and two ground rails (or one wide ground rail) in the middle for the two N rows).

2. For mixed-signal blocks, it may be necessary to draw two sets of power and ground rails, one for analog and one for digital. Follow the same procedure for drawing the two sets of power and ground rails.

Creating Wells

1. Depending on the process technology, you might be required to draw an explicit n-well over areas where PMOS transistors are to be placed. Change the drawing layer in the LSW to n-well drawing and create a rectangle around the p-transistor area.
2. For a twin tub process, a p-well (or wellbody) will have to be drawn in a similar manner around the PMOS devices.

   Note: For the sample pcell library the individual PMOS pcells have n-wells already built around them. However, in order to prevent DRC violations being reported for n-well to n-well spacing as well as to ensure correct hookup of the bulk contacts, it is advisable to build an explicit n-well around all the p-transistors.

Creating Guard Rings

Guard rings should be drawn around sensitive circuitry or around the entire block, depending on how sensitive the circuit is expected to be. An N guard ring (primarily for p-devices) is drawn as an n-diffusion path inside the n-well, tied to vdd, whereas a P-guard ring (primarily for n-devices) is drawn as a p-diffusion path in the p-substrate, tied to vss. If there is only one diffusion layer, it may be necessary to create n- and p-diffusion using the appropriate implant layer

The guard rings can either be created using the *Create – Polygon* command or by using the '*Create – Multipart Path* (MPP). A multipart path is a path comprising of a master path (in this case, diffusion) and one or more sub-parts (in this case, the lowest metal layer in which the vdd/vss have been drawn and contacts to hookup to vdd/vss).

Follow the steps given in order to make Virtuoso XL recognize the guard ring connectivity below to create a guard ring using the *MPP* option;

1. Before creating the multipart path you must turn on the *Sticky Net* option in the Layout Editor Options form for Virtuoso XL to recognize the guard ring connectivity.

   This option assigns the `lxStickyNet` property to an MPP object. This property can be edited from the Edit Properties form.
2. Choose *Create – Multipart Path* and the Create Multipart Path form appears.
3. In the LSW set the drawing layer to ndiff for a N guard ring (for a P guard ring change the drawing layer to pdiff). If needed change the width, offset, etc. so that DRC requirements will be met. Make sure the *Choppable* option is not turned on. Specify a user-defined template name.

4. At the bottom the Create Multipart Path form choose *Subpart*. The ROD Subpart form appears.

5. If there is only one generic diffusion layer, the appropriate implant layer around the diffusion will need to be added. In the ROD Subpart form turn on the *Offset Subpath* option. In the *Layer* cyclic field choose the N implant layer for N guard ring (for a P guard ring choose the P implant layer). Change the right offset so that there is no DRC violation. Turn off the *Choppable* option. Choose *Add* and the subpart is added to the list box.

   Note: The diffusion layer should never be chopped because it would break the guard ring. The *Choppable* option in the MPP form should therefore be turned off for the diffusion layer, and turned on for the metal layer.

6. Turn on the *Enclosure Subpath* option. This corresponds to the metal layer for power hook-up. Change the *Layer* cyclic field to the appropriate metal layer. The *Choppable* option can be used to allow wires to connect to pins inside the guard ring.

7. Set the connectivity information for the sub-path by changing the *Connectivity* option to *Net*. Change the net name to vdd and click *Add*.

   Note: Subpaths with the *Connectivity* set to *Pin* do not translate correctly in the router environment. If the *Connectivity* type is set to *Net*, make sure it is hooked up to the appropriate pin/rail to extract the correct connectivity.

8. Choose the *Sub-rectangle* option which corresponds to the contacts for connecting the main diffusion path with the metal sub-path and provide necessary information regarding enclosure, spacing etc. Click on *Add*.

9. Click *OK* in the ROD Subpart form.

10. In the layout window create the guard ring around the desired area.

11. Use the *Edit – Other – Chop* command to chop the portions of the sub-path (the metal layer) where the pins have to be accessed from outside.

Saving and Loading the Multipart Path

1. The multipart path information can be saved to an ASCII file or the technology file. In the Create Multipart Path form click *Save Template* and the Save Multipart Path Templates form appears. This loads the MPP template into the technology file. Then you do not have to save the technology file in order to save this template.

2. Specify the Templates to be saved. If the technology file cannot or should not be modified due to design and control reasons, then the template should be saved to an ASCII file. If you save the template to the technology file then the template file name will be listed under the *MPP Template* cyclic field on the Create Multipart Path form.

Note: Even if the technology file was not modified and you plan on discarding the edit, you will still need to have write permission for the technology file in order to save the MPP template.

3. If you want to load ASCII template files into the Create Multipart Path form choose *Load Template* in the Create Multipart Path form. In the Load Multipart Path Template form specify the template name.

Using Guard Rings with the Virtuoso XL Layout Editor

1. After the guard ring has been created it can be hooked up to the appropriate supply rails/ pins. This should be done manually in Virtuoso XL or interactively with the Virtuoso custom placer.
2. If the Virtuoso custom placer is being used for a row-based placement of devices, it is possible to generate a power template by predefining the metal layer, width, net name and alignment (inside, outside or center with respect to the component rows) for the P rows and the N rows.
3. The power template, along with the I/O pins and bounding box information, can be save to an ASCII template file using the *Design – Save Design To Template*. You can use the ASCII template for trying out different layout options. This template may also include spares and feedthrus (in case such top-down information is available). This plan can serve as an abstract, if required, for the top-level floorplan.
4. If a design is instantiated in an existing layout, it should be flattened using *Edit – Hierarchy – Flatten* command. Turn on the *Preserve Pins* option in the Flatten form so that connectivity information is not lost.

   Note: Before you flatten the design you will need to choose the components in the layout window. Choose *Connectivity – Propagate Nets* command and click on *Set All Net Names to Terminal Names* (assuming that pins have already been generated in the layout and there is a net to terminal correspondence between the layout and the template), in order for connectivity to propagate to the top-level. This is due to a current tool limitation that does not retain the pin names when a cell is flattened.

Pin Placement

Pin Placement and Constraint Specification

The first step in device placement is to create a rough floor plan of the design. This involves placing the I/O pins and choosing the row/columns patterns either manually or using the automated placer (Virtuoso custom placer). You must know how many rows of PMOS/NMOS devices or standard cells are going to be required for the given block size or aspect ratio. Analog/mixed-signal blocks require determining the location and placement of the passive devices (resistors and capacitors). The overall plan should focus on minimizing routing.

Pin Placement

Typically, the I/O pin placement is based on a top-down floor plan/ block template. The pins can be placed either manually (Virtuoso XL) or by using the *Place – Pin Placement* command (Virtuoso custom placer). If the pins need to be placed inside the block (not on the edge) as ports, they should be manually placed in Virtuoso XL. If the pins need to be aligned to the periphery of the block then use the *Place – Pin Placement* command.

The Pin Placement form allows you to;

- align pins to any boundary edge (including non-rectangular boundaries).
- assign specific pin ordering for a group of pins.
- place a group of pins with any given spacing between each pin.
- expand/collapse iterated bus pins and report pins and their assigned constraints.

Note: Pins constrained through the Pin Placement form are treated and translated as a constraint. These pin constraints are maintained by the Virtuoso custom placer as well as Virtuoso XL in constraint assisted mode.

1. Choose *Place – Pin Placement* from the layout window.

   Note: You must have created a prBoundary before invoking the *Pin Placement* command.

2. Choose a pin displayed in the list box and you can constrain the pin the an edge, specify the order, specific the spacing between pins, and if you want the pin to be floating, or protected after placement. At the bottom of the Pin Placement form click *Apply*.

   Note: The pin has been placed on the edge of the prBoundary is no longer in the list box. Only unplaced pins are displayed in the list box because the *Display* is set to unplaced.

3. To display the placed pin change the *Display* cyclic field to *Placed*. The list box updates to show you the pin that has been placed and the placement information for the pin.

4. Vertical or horizontal rails may be created for multiple pins on the same net (for example, for power, ground, feedthrus, spares etc.). Choose the pin(s) in the list box and click on either Hrail or Vrail. Because the railed pins stretches the entire span of the prBoundary, no other pin on the same layer may have previously been placed on that span.

5. Pins can be ordered for a specific edge. Choose a pin(s) and in the *Order* text field specify a number for the order. The lower the number the higher priority that the pin will be placed first. You can use the arrow keys in the *Change Order* box to change the order of the pins. The *Swap Order* button swaps the order of two pins.

6. In the *Pitch* field type in a spacing between two pins or a group of ordered pins.The pitch is center to center spacing. If the pitch if not specified for a group of pins, then by default they will be placed with equal spacing, distributed along the selected edge. Change the *Location* cyclic field to *Fix at Placed Location*. The pins will not be moved from this location when running the Virtuoso custom placer.

7. To keep pins in their exact location when running the Virtuoso custom placer change the *Location to Fixed* at *Placed Location*. Protected pins can be moved to another location. The placer will keep the protected pin at this new location.To unfix or unprotect a pin change the *Location to Floating*. Fixed and Protected pins must be placed inside of the prBoundary. The *Fix after Placing* option causes the pin to be placed at a location determined by the placer.

8. You can specify that iterated pins are shown expanded or collapsed. They can be expanded or collapsed at any time. Collapsed iterated pins are treated just like any individual pin and will get spaced uniformly in the given boundary edge unless a specific pitch is provided by the user.

9. The pin placement information is stored in the layout cellview. You may want to create different placements of pins and have the information stored in ASCII template files. Use the *Design – Save Design To Template* command to create the ASCII template files. In the Template File form specify in the *I/O Pins* section *Locations* and *Constraints*. Specify a Template file name. Once you click *OK* or *Apply* the template file name is saved to the layout cellview. When you open the Pin Placement form and click on Template file the template file name is loaded in the *Name* field in the Load Template form.

Constraint Manager

1. Constraints may be applied to instances and nets using *Tools – Constraint Manager*. Constraints can be added either to the schematic or to the layout. Choose the cellview to add the constraints in the *Select Cellview* cyclic field. Use the *Edit – Other – Lock*

Selected command to lock critical devices to ensure that critical placements are not disturbed during routing and other subsequent iterative phases. Locked devices may be unlocked during the course of the design using the *Edit – Other – Unlock Selected* command.

Note: Constraints can be entered to the schematic or layout cellviews. Constraints entered in the layout will be overwritten whenever layout generation is re-run. Schematic constraints will be applied each time a layout is generated. You can save constraints for any given cellview in to a file and load them back again to any cell view (described later).

2. Choose *Options – LayoutXL* and turn on the *Constraint Assisted Move* and *Stretch* option to maintain the constraints in Virtuoso XL. When there are any changes in Virtuoso XL violations will be prevented if the constraints have been met in the current layout. Constraint assisted mode will not ensure that the constraints are met when changes are made in the layout (except for *fixed* or *lock* constraints).

3. Geometric constraints are automatically passed to the placer when the placer is used to generate automatic placement. The placer tries to maintain these constraints but there is no easy way to verify if the constraints were met. For more information about constraints supported by the placer, refer to the to "Using the Virtuoso Custom Placer" in the *Virtuoso XL Layout Editor User Guide*.

4. The Virtuoso custom router only supports net-based constraints. Net-based constraint data is sent to Virtuoso custom router by an ASCII `.do` file called *`cellName`*`Constraints.do`. The custom router automatically loads files named *`cellName`*`Constraints.do`.

5. The Virtuoso XL wire editing environment supports net-based constraints and follows the constraints set in the constraint manager once the constraints are saved to the cellview.

6. The constraints can be saved to an ASCII file within the Virtuoso constraint manager. The ASCII file can be modified if required, then loaded again. In the Virtuoso constraint manager form choose *File – Export*. In the Export ASCII Constraint Entities form specify the type of constraints to save and the cellviews.

7. To try out different floor plans and placements, first save pin and/ or instance placement and load them later using the *Design – Save Design To Template* command.

8. The placement style and design requirement dictate whether the next step in placement should be done using "Assisted Manual Placement in the Virtuoso XL Layout Editor" on page 203 or "Automated Placement using the Virtuoso Custom Placer" on page 208, either partially or entirely. If the design does not require a custom placement and comprises mostly of complementary MOS devices and standard cells, then the design can make use of the automatic placement functionality provided by the Virtuoso custom placer. For more manual custom placement Virtuoso XL is the proper tool to be used effectively for a wide range of design scenarios.

9. Whichever process is used to perform the Placement task the next step would be "Post Placement Steps in the Virtuoso XL Layout Editor" on page 202.

Post Placement Steps in the Virtuoso XL Layout Editor

Once the placement has been completed it is a good idea to run DRC to capture and fix any problems prior to sending the placed design to the router (such as spacing violations).

> **Note:** Typically the DRC rule decks include checks for latch-up, and violations are flagged if substrate/ well taps are not added to meet the process requirement. Many rule decks have an option to turn off this check to prevent a large quantity of errors when DRC is run prior to adding these taps. Even if this is not explicitly checked by DRC, it is a common practice to add these ties based on some general design guidelines. This can either be done now or after routing has been completed.

1. Taps are usually provided in the pcell sample library as symbolic contacts, such as *metal1* to n-well contact (for p-devices) or *metal1* to substrate contact (for n-devices). The taps should be instantiated so that there is at least one within the maximum allowable radius of the gate of each MOS device as per the design/process requirement. The well taps for a given set of p-devices must be placed within the same well. The taps should be placed so that there is minimal hook-up required, (i.e close to the rails or hooking up directly to the source of the adjacent MOS device that connects to the appropriate supply).

   Note: Taps/ties involve layers such as wells, implants etc. that should be translated as reference layers to the router, not as routing layers. If the taps are placed prior to sending the design to the router, the Virtuoso custom router will flag them as incomplete nets because the associated layers are reference layers with implied connectivity. The Virtuoso custom router would not be able to route to the taps because they are not on actual routing layers. The placement and interconnect of ties should be done in Virtuoso XL after all the nets have been routed in the Virtuoso custom router.

2. You will need to create rails in Virtuoso XL for strapping legged/ folded devices using the pin to trunk or grid routing capability in the Virtuoso custom router. Assign appropriate connectivity to these rails/shapes for the Virtuoso custom router to recognize the nets.

Assisted Manual Placement Detailed Instructions

Assisted Manual Placement in the Virtuoso XL Layout Editor

Follow the steps below if you have already generated the components in your design using the *Design – Gen from Source* command.

Manually Placing Devices

1. Use the *Edit – Move* command to place the p-devices close to the power rail and the n-devices close to the ground rail. To make selection of devices easier, turn off selectability of the boundary and pins in the LSW.

   Note: If you cannot view the flight lines as you move the devices then turn on the *Display Draglines* option in the Move form. The environment variable flightLineEnable controls the display of draglines or flight lines in the layout window. You can set this environment variable to true in the `.cdsenv` file.

2. Another way to view flight lines is by using the *Connectivity – Show Incomplete Nets* command to view the flight lines. The flight lines can be used as guides for optimally placing the devices to minimize routing between the devices as well as to the I/O pins.

   Note: If you cannot view the flight lines as you move the devices, then turn on the *Display Draglines* option in the Move form. The environment variable flightLineEnable controls the display of draglines or flight lines in the layout window. You can set this environment variable to true in the `.cdsenv` file.

Manually Aligning Devices

If the *Transistor Chaining* option was set in the Layout Generation Options form, the complementary NMOS and PMOS transistors are automatically aligned vertically. If the *Transistor Chaining* option was not set in order to have more manual control, the complementary p- and n-devices should still be vertically aligned to minimize routing. The connectivity lines between devices should be used as guide for tracking and placing the complementary devices.

Devices to be placed in the same row should be aligned horizontally and complementary P and N devices should be aligned vertically. Use the *Edit – Other – Align* command to align devices with DRC correct spacing.

1. Choose the devices to be aligned

2. Change the *Align Using* cyclic field to *Layer BBox*.
3. *Reference Point* to *centerCenter*.
4. *Alignment Direction* to *Horizontal*.
5. The Layer (for example, p-implant or n-well for p-devices).

   Note: To vertically align complimentary n and p devices, you could use the *poly* layer with centerCenter reference.
6. Apply Spacings to Component Pitch and set the Pitches.
7. Sort Components By Align Direction Order
8. Click *Set New Reference*. You are prompted to enter the origin for alignment. In the layout window choose the origin of one of the selected devices and the devices are aligned.

   Note: If the auto-spacing rule is defined for the pcells, then devices when brought close to each other can be made to snap to an alignment position even if they do not share diffusion.

Placing Devices from the Schematic

If the devices have not been generated from the Layout Generation Options form, then use the *Create – Pick from Schematic* command to place the devices.

1. Choose *Create – Pick from Schematic* to select devices from the schematic.
2. Turn on the *Transistor Chaining* and *Transistor Folding* options for MOS devices.
3. Specify the NMOSMaxWidth and the PMOSMaxWidth (i.e., width of the device in microns (u)).
4. Turn on the *Draglines* option to determine relative placement of the devices.
5. Choose *Unplaced* and the Pick from Schematic Instance/Pin List box appears.
6. Choose the devices to be placed.
7. Click in the layout to place the devices.

Transistor Folding and Chaining

Some of the wider devices might already have multiple fingers in the schematic, in which case the corresponding layout device will also be legged. However, from a floorplan consideration,

some of the devices might still be wider than the rest so that it creates inefficiency in placement and might create alignment problems. These devices can be folded interactively.

1. Choose the *Edit – Transistor Folding* command and specify the *Number of Fingers*.

   Note: Multiple finger devices in the schematic or in the layout can not be folded.

2. If you known ahead of time what the maximum row width should be, then the devices can be automatically folded based on a cut-off not exceeding the maximum row width. Choose the *Design – Component Types* command. In the Edit Component Types form set the *Fold Threshold* from the *Edit Parameters* cyclic field.

3. The devices on the same net should be abutted for maximum sharing of source and drain to save area. If auto-abutment properties have been added to the devices in the pcell code, then the devices that can share source/drain should automatically abut when one device is brought close to the other device. Choose *Options – Layout XL* and turn on the *Auto Abutment* and *Auto Permute Pins* options for abutment to take effect.

   Note: If the *Transistor Chaining* option is turned on in the Layout Generation Options form, (even if *Transistor Folding* is not turned on), then the devices will be abutted.

For more information about transistor folding and chaining with the Virtuoso XL layout editor, refer to the Using Transistor Folding and Using Transistor Chaining sections in the *Virtuoso XL Layout Editor User Guide*.

Ignoring Devices

1. For an analog layout, it may be required to place dummy gates/resistors for matching requirements. For Virtuoso XL to understand and interpret the connectivity correctly, it is necessary to add the property lvsIgnore to the dummy instance. This will prevent Virtuoso XL from flagging the dummy gates/resistors as a mismatch.

Cloning Structures

If there are some repetitive, identical topologies in the design (for example, symmetrical structures in layouts of analog blocks such as mixers or bit slices/datapath structures), they can be replicated in the layout using the *Create – Clone* command. Cloning can understand and create connectivity for the target while maintaining the same structure as the source.

The conditions for cloning are that the source and the target should be isomorphic (have identical topology), they should have identical parameters, and they should have the same instance masters.

1. To replicate a structure in the layout, choose *Create – Clone*.
2. In the schematic window choose the devices to be cloned.
3. In the Cloning form make sure that the *Target Schematic* and the *Target Layout* are set to the correct cellviews.

   Note: The target structure (components and interconnects) should not exist in the layout view at the time of cloning (should be deleted prior to cloning if generated already). Otherwise the cloning command will generate message stating that the target structure is incorrectly or partially implemented.
4. To find the target matches turn on the *Entire Target Schematic* option and *Permutation* and *Exact Match*.
5. Click *Find Matching Targets*.
6. The unplaced devices will be listed in the *Unplaced* list box.
7. Choose the devices from the *Unplaced* list box and place them in the layout window.

   Note: If the structure to be cloned has been routed (has interconnects), then the structure should be selected from the layout view (i.e., both the components and the interconnects should be selected). Selecting the source from the schematic will only clone the components and their relative placement, and not the routing/ interconnects.

Multiplication and Series-Connection Factor

You may be required to multiply an instantiate of a schematic component, without actually drawing it multiple times in the schematic.

For information about the sfactor, refer to the Series-Connection Factor (sfactor) section in the *Virtuoso XL Layout Editor User Guide*.

1. Add the mfactor property (multiplication factor) as a CDF parameter, or as a property of an instance or a cell. Components having mfactor greater than one are connected in parallel in the layout. In the Layout XL Options form the *Generate Multiple Instances* option must be turned on.

2. You may be required to draw a single large component in the schematic as a series of smaller components (for example, to ensure matching in a large resistor). This can be done by setting the sfactor (series factor) property either as a CDF parameter or as an instance property.

   The `lxSeriesTerms` property must be defined on the instance in the schematic in order for Virtuoso XL to understand the connectivity of the devices with sfactor >1 and instance terminals>2. In the description below of setting the lxSeriesTerms PLUS and MINUS are the 2 instance terminals.

   lxSeriesTerms = “PLUS MINUS’”

   In the Layout XL Options form the *Generate Multiple Instances* option must be turned on.

   For more information about the mfactor, refer to the Multiplication Factor (mfactor) section in the *Virtuoso XL Layout Editor User Guide*.

   The sfactor property can be applied to two or three terminal resistors, capacitors, and inductors. If both sfactor and mfactor are defined for a device, then sfactor is ignored.

   Note: The sfactor and mfactor properties are a subset of a more general concept of one-to-many, many-to-one, and many-to-many mappings. These mappings allow flattened layout devices to correspond to a high-level schematic instance. Or a high-level layout instance can correspond to multiple schematic pieces, or a combination of both.

Completing the Placement

1. Sometimes it may be necessary to stretch the contacts in the layout to accommodate routing of critical nets. If the pCell code for the MOS devices have stretch handles assigned to the contacts, they can be stretched using *Edit – Stretch* command. You can also stretch the gates of the pcells if they are assigned stretch handles in the code, but this is usually not done if the layout is being driven by the schematic.

   For more information about stretchable pcells, refer to the *Virtuoso Relative Object Design User Guide*.

2. Once the placement steps are completed the next task is to route the devices. However there are some recommended Post Placement Steps in the Virtuoso XL Layout Editor that should be performed first.

Automated Placement Detailed Instructions

Automated Placement using the Virtuoso Custom Placer

Components can be placed in an automated manner using the Virtuoso custom placer within the Virtuoso XL environment. The Virtuoso custom placer can be used to perform “what-if” floorplanning/placement analysis and used as the starting point for final placement.

Data Preparation

Component types are used to identify MOS devices for chaining and folding, and to identify groups of devices for row-based placement.

1. Choose *Design – Component Types*.
2. In the *Type* field specify the user-defined name and then click *Add*. The new type will be added to the cyclic field. A type can be deleted by selecting the type and clicking on the *Delete* button.
3. Assign a device (s) to a specific component type. Choose a device in the list box and click on the arrow key to move them into the *Components in Type* list box. You can also select a device in the layout window and then click *Add Selected Components*. The selected device is placed in the *Components in Type* list box. To delete components from the *Components in Type* list box click on the *Delete Component* button.
4. Specify the *Component Class* from the cyclic field at the bottom of the form (*Undefined*, *PMOS*, *NMOS*, *STDCELL*, *STDSUBCONT*, or *FILLER*).
5. For chaining and folding to occur you must specify values for *Width Par Name*, *Fold Threshold*, and *Active Layer* (change the cyclic *Layer* field for the active layer). The *Source Terminal*, *Drain Terminal*, *Gate Terminal* and *Bulk Terminal* parameters are optional. The *Fold Threshold* should be specified in microns, (for example, 5u.)
6. If pins have not already been placed, use the *Place – Pin Placement* command to place the pins. See the pin placement step in the floor plan section for details.

   For more information on how to use the Virtuoso custom placer for automated placement, refer to the “Using the Virtuoso Custom Placer” chapter in the *Virtuoso XL Layout Editor User Guide*.

   Note: You may want to constrain the pins by the using the Virtuoso constraint manager or by fixing the pins (to generate temporary fixed constraints) as described in the pin placement step.

Partitioning the Design

To plan the layout more efficiently you can assign components to parts of the layout area. This allows you to monitor the placement more closely. You will create partitions and assign components to the partition. The placer will keep the devices inside the partition.

1. Before you create a partition, you must create a shape that will be attached to the partition. In the LSW add the softFence drawing layer purpose pair to the list of Valid layers.
2. Draw a rectangle on the softFence layer around the NMOS devices that you want to keep together during placement.
3. To create a partition choose *Place – Partitioning*.
4. Click *Link* to link the Partitioning form to the cellview. In the CIW a message appears telling you that the Partitioning Form is linked to the design “VCPLIB BUF layout”.
5. By default the *Partition name* is set to *Boundary*. To create a new partition clear the text field and type in a user-defined partition name and click on *Create Partition*.

   The new partition name appears in the cyclic field with a question mark next to the name. The question mark denotes that the partition does not have a shape attached to it.
6. To attach the softFence rectangle to the partition choose *Attach Shape* and choose the rectangle.
7. To add the devices to the partition change the Target Partition to the new partition name and choose the devices in the *Component Name* list.
8. Choose *Move* to move the devices into the partition.

Planning Placement for Standard Cells

If the design requires row-based placement of Standard Cells, then follow the steps given below:

1. Set the component type parameters lxMOSDeviceType for STDCELL for all the standard cells.
2. Choose *Place – Placement Planning* and change the *Style* to *Assisted Standard-Cell*.
3. Specify the Partition to be used.
4. For an estimation of rows, choose *Row* and specify the *Row Name*, *Number of Rows*, *Utilization*, and *Direction*. Choose *Calculate Estimates* and the minimum rows

needed (default setting), number of rows, horizontal utilization, vertical utilization and row spacing.

5. Specify an area in which rows will be generated by selecting Region. Choose *Draw* and provide the coordinates or draw a rectangle in the layout. If a region is not provided then the entire process boundary area will be used.

   Note: Currently, a region may need to be drawn even if the entire process boundary is going to be used in order to make sure that devices are not placed on top of pins because there is no automatic cut-outs for the pins.

6. Once the factor by which to estimate the placement and the placement area has been set, click the *Calculate Estimates* button to run incremental layout generation.

7. If the estimated numbers are acceptable, complete the setup for row generation by definingRails (such as layer name, width and rail pattern). Choose *Update Layout*.

8. Change the *Style* to *Manual User-Defined* and the Placement Planning form updates to show the rows that have been generated.

9. To run the placer go to step “Running the Placer”.

Planning Placement for Assisted CMOS

If the design requires row-based placement of MOS devices, follow the steps given below:

1. Set the parameters lxMOSDeviceType for PMOS/NMOS devices. The lxDeviceWidth and lxMaxWidth parameters are required to evaluate folding parameters.

2. Choose *Place – Placement Planning* and change the *Style* to *Assisted CMOS*.

3. Specify the partition to be used.

4. To obtain an estimate of the placement choose *Row* and change any one of the following row options (*Row Name*, *Utilization*, and *Number of Rows*).

5. Specify an area in which rows will be generated by selecting *Region*. Choose *Draw* and provide the coordinates or draw a rectangle in the layout. If a region is not provided then the entire process boundary area will be used.

   Note: Currently, there may be a need to draw a region even if the entire process boundary in used, in order to make sure that devices are not placed on top of pins because there is no automatic cut-outs for the pins.

6. Choose *Rails* and define the power and ground layer, width, and net name. Set the *Rail Pattern*.

7. Choose *Components* and specify the PMOS and NMOS *Width Threshold*, *Diffusion Spacing*, and *Alignment*. Click *Calculate Estimates* to run incremental layout generation.

   Note: If the supply rails are in higher layer metals (*metal2* and above), the devices should be aligned to the supply rails to minimize power routing (i.e., component and power alignment should be identical, both Inside or both outside). If the supply rails are in *metal1*, devices should be aligned opposite to the rails to prevent shorting (i.e., if component alignment is inside then power alignment should be outside and vice versa).

8. If the estimated numbers are acceptable and the row definitions and overall placement style are in accordance with the design requirement choose *Update Layout*.

   Note: When you choose *Update Layout* the *Layout Generation* option is run on the design. Specify if you want to Regenerate All devices, or only missing components.When Layout Generation is run you can also specify that constraints on objects are preserved by selecting the *Preserve Constrained Objects* button.

9. Change the *Style* to *Manual User-Defined* and the Placement Planning form updates to show the rows that have been generated

10. To run the placer go to the step "Running the Placer".

    Note: If the rows generated do not fit within the prBoundary and the placer is run, the placer will not behave correctly. Make sure that the rows are within the prBoundary either by stretching the boundary or by readjusting your rows. If the boundary is stretched run Update Layout in the Placement Planning form. The pins will be realigned to the new boundary.

Planning Placement for Manual User-Defined

1. If the design requires customized rows, follow the steps given below:

   a. Bring up the Placement Style form by clicking on *Place – Placement Style* from the Virtuoso XL window. Click on *User Defined* under *Row Definition*.

   b. Choose name (**optional**), direction, width/height, number of copies (i.e number of identical rows based on this specification), spacing etc. for the rows along with the properties of the components that would go into the row(s).

   c. Choose *Create New* to see the rows get generated. Any or all of the rows can be selected, properties modified, and then a click on *Update Rows* will update the row properties at any time. Choose *OK* or *Apply* to write the row definitions to the cellview.

   Note: The *User Defined* mode sends the devices 'as is' to the placer. Hence, if it is

desired to fold/ chain any or all of the devices, this should be done up front prior to defining the rows. This is different from the *Component Assisted* mode where the devices are regenerated with folding/ chaining options, before being sent to the placer.

d. Go to the step "Running the Placer". However, if the generated rows do not fit within the prBoundary and the placer is run, it will not behave correctly. Make sure that the rows are within the prBoundary either by stretching the boundary or by readjusting your rows. If the boundary is stretched and Apply or *OK* is clicked in the pin placer the pins will get realigned to the new boundary.

2. If the design requires area based placement of the devices (for example, AMS type designs), follow the steps for creating and saving constraints as described in the floor plan section. See "Constraints" on page 151 for more details. Directly run the placer (go to step "Running the Placer") without going through the steps for defining placement style (area based placement is the default behavior of the placer).

Note: If chaining and/or folding has been used the resulting chains will be treated as one object by the area placer. It is possible that the chaining and folding can create an object that is too long for the PR boundary which will cause the auto placer to fail to place this object. Long chains should be broken into reasonable lengths before sending them to the area placer.

3. The recommended method of placement that yields the best results is to place the complementary MOS devices/standard cells in rows and manually place the remaining devices (for example, resistors, capacitors and critical devices, if any). Follow the steps below:

a. Define rows either manually in *User Defined* mode or automatically in *Component Assisted* mode for placing the devices that need to go into rows.

b. Create room for manually placing the remaining devices by adjusting the height/ width of the rows, if required (these extra devices can be placed in this area either before or after running the placer to place the devices that need to go in to the rows).

c. Run the placer as described in the next step but making sure to select all the devices that need to be placed by the automatic placer and choose the *Place Selected Objects Only* option in the placer form.

Running the Placer

1. Once the placement style and areas have been defined choose *Place – Placer*.

2. Choose *Group CMOS Pairs* if it is important to maintain grouping of CMOS devices that have been generated with chaining. This option does not work with devices that have been manually abutted.

3. Choose Group M factor devices to group complimentary MOS devices that have a mfactor (multiplication factor).
4. Choose *Place Selected Objects Only* to place only selected objects.
5. Turn off the *Allow Rotation* option to override defined allowed rotations and temporarily fix rotations for that placement run.
6. Choose *ECO Mode* for incremental update only. This mode will not disturb the original devices and will put in new devices as needed.
7. Choose the placement algorithm, *Global Placement* and/ or *Optimize Placement* (detailed placement).

   Note: Global Placement should be run before any detailed placement passes because it tries to minimize the net lengths irrespective of the initial placement. On the other hand, detailed placement tries to refine an existing placement. The recommended setting on the placer form is to have both *Global Placement* and *Detailed Placement* options turned on (default settings).
8. Set the *Runtime* option to quick, moderate or optimized. These options have different degrees of wire length reduction depending on the design style. It is a trade off between runtime and quality.
9. The Run Row Spacer within Rows and between Rows will compress or expand the rows. The placer considers 0% just enough room for routing while 10% (maximum setting) gives an additional 10% routing resource.
10. A Rules File can be supplied that is specific for placement or it can be the same icc rules file for routing in the Virtuoso custom router. Often better results can be obtained by using a Virtuoso custom placer specific rules file. By default, the Virtuoso custom placer uses the technology file information.
11. For standard cell designs, choose *Filler Cells* if the row definitions include filler cells to fill all the gaps between your standard cells with filler cells.
12. The *Save As* option redirects the results to another cellview. If this is option is not selected then the Virtuoso custom placer results will automatically import to the current cellview.
13. Placement is iterative so it is possible to step through the entire placement sequence by pre-placing and locking critical devices, then running row-based placement for MOS devices and/or standard cells, and finally running area based placement for the rest of the design.

    Note: .Because the Virtuoso custom placer permits placement generation in minimal time, it is particularly suited for doing floorplan/placement analysis. Different constraints can be set each time and several iterations run until satisfactory results are obtained. An

existing placement can be refined by running detailed placement only (*Optimize Placement* option in the Auto Placer form), with different initial placements as seeds. High quality placement may be achieved in this manner by making minor manual adjustments based on top-level floorplan requirements or other high-level constraints.

14. Once the device floor plan and placement steps are completed, the next task is to route the devices. There are some recommended Post Placement Steps in the Virtuoso XL Layout Editor that should be performed first.

For more information on using the Virtuoso custom placer see the *Virtuoso XL Layout Editor User Guide*.

Translate Design Detailed Instructions

Create the Rules File

You must provide a rules file for translating the design from Virtuoso XL to the Virtuoso custom router. To create a new rules file choose *Route – Rules – New Rules*. To use an existing rules file choose *Route – Rules – Open Rules*. The translation rules can be standard for most designs and reused each time the translation is turned on. An example translation rules file is provided in this document and can be referenced as needed from the section descriptions.

1. Choose *Route – Rules – New Rules* to create a new file or *Route – Rules – Open Rules* to modify existing file and follow the steps to create and modify a rules file.
2. In the Rules form choose *Layer* and click on *Add*. The Add Layer form appears with the list of available layers. Choose the layer purpose pairs that are to be used for routing (appropriate metal layers and *poly* layer).

   Note: The layers that show up in the list are the ones that have been set as valid layers through the LSW or the technology file.The section in the rules file is called "iccLayers =" which defines the routing layers
3. In the Rules form specify the *Layer Function*. The diffusion layers may need to be added if pins have been defined on these layers (typically drain and source are defined on the diffusion layer for autoabutment).

   Note: In case there is only one diffusion layer in your design, choose the layer function as either N Diffusion or P diffusion. The translator does not understand a generic diffusion layer.
4. Set all of the routing layers to T (translated). Do not turn on the *Ref Layer* option for the routing layers. The diffusion layers should be both T and Ref (translated and used as reference) because the drain and source contacts have been defined on the diffusion layer for abutment purpose.

   Note: Because there are bulk contacts defined on the nwell layer (for the PMOS pcell) and on the wellbody layer (for the NMOS pcell), you may be required to translate these layers and declare them as reference layers (turn on T and Ref for both) so that the router understands the connection for substrate and well taps.
5. Set the routing directions for each layer to either (vertical, horizontal, orthogonal or off).
6. Arrange the layers in ascending or descending order by providing the Palette Order number for each layer.

7. Choose the *Vias* option and click *Add* to define the vias for via array resizing and custom enclosure rules. Choose the symbolic vias and contacts from the technology library. Check that the layers definition (layer1, layer2 and via) is correct. This information creates the "iccVias =" section which defines the vias for via array resizing and custom enclosure rules.

8. Choose the *Equivalent Layers* option and click *Add* to define layer purpose pairs which have equivalent connectivity meaning. The drawing, pin and net purposes for each layer should be declared as equivalent. If there are more than one *poly* layer, those should be defined as equivalent as well. The "iccEquivalentLayers =" section contains this information.

9. Choose the *BoundaryLayers* option to define the external boundaries which limit the extent of the routing layers in the Virtuoso custom router. The default layer is prBoundary (purpose by). The default spacing is 0.0 which means the routes can run right to the edge. This information is stored in the "iccBoundaryLayers =" section of the translation rules file.

   Note: Each layer can have a unique range as per the technology file in the case where 1/2 DRC rule is required to allow the cells to abut. To modify the design rule in the technology file use a syntax similar to the following:

   ```
   list ('("M1" "drawing") '("prBoundary" "drawing") techGetSpacingRule
   (techGetTechFile(ddGetObj("tsmc18")) "minSpacing" "M1") / 2.0)
   ```

10. Choose the *Keepouts* option to define the regions where no routing is allowed. Logical functions can be used for example to restrict routing over MOS gate regions. This information appears in the *"iccKeepouts ="* section of the translation rules file.

11. Choose the *Conductors* option to define any special conductors other than those defined by the routing layers.

12. Choose *File – Save* to save the rules file.

For more information on this subject check the *Virtuoso Custom Placement and Routing Preparation Guide* and click on *Translation Process Overview*.

Interactive Routing capabilities

Typically if you work with blocks with a limited number of devices, a combination of Virtuoso XL and Virtuoso custom router can be used for device-level layout and routing. A common use model is to create a layout in Virtuoso XL and to route all or part of the design interactively. Instead of having to translate the design and then route either interactively or automatically in Virtuoso custom router you can use the wire editing capabilities of Virtuoso XL to provide a correct-by-construction editing capability that includes a graphically assisted mode for DRC correct interactive routing.

Prerequisites

In addition to the Virtuoso XL technology file requirements you will need to define which layers and vias are to be used for routing. This information is defined in a rules file. For more information about rules files, refer to "Translate Design Detailed Instructions" on page 215.

Enable Wire Editing

1. In the layout window select *Options – Layout XL.*

   Note: Virtuoso XL must be turned on to enable wire editing.

2. Turn on *Enable Wire Editing* and click OK.

   Virtuoso XL wire editing commands are available from the pulldown menus.

Setting Constraints

Before routing, you can set constraints on nets with specific requirements. You can set constraints by loading a .do file or through the Virtuoso constraint manager. For more information about setting constraints in the constraint manager, refer to "The Virtuoso Constraint Manager" on page 151.

Constraints and Definitions Set in a .do File

When the wire editor is enabled, the `.do` file is used for setting constraints and definitions rather than a script to drive a batch job, as in the Virtuoso Custom router. To create and load a `.do` file, follow these steps:

1. Create an ASCII file using any text editor. The `.do` file can include the following four commands.

- circuit
- define
- rule
- do

For information about syntax, refer to the *IC Shape-Based Placer/Router Documentation*.

2. Select *Options – Route*.
3. Turn on *Do File*, type in the name and path of the `.do` file and click *OK*.

Creating Interconnect

You can route either a single wire or multiple wires between connectivity objects. While routing, you can insert vias to change the routing layer and push aside routes and components, depending on the options you have set.

You can use the middle button Layout pop-up menu to access routing options while in an active command or choose *Options — Route* from the pull down menus. For details about setting up the routing environment and controlling the routing style, refer to the *Virtuoso XL Layout Editor User Guide*.

1. Select *Create – Path*.

   Note: To create multiple paths, select *Options — Route* and make sure *Enable Bus Routing* is must be turned on in the Bus tab. The default setting for *Enable Bus Routing* is on.

2. Click or area select connectivity objects (pin, via, or an existing path).

   Flight lines appear showing the connectivity of the selected nets, and the status bar shows the name of the selected nets.

3. Click points where you want to route the paths.

   While you are routing, a preview of the active path segments stretches from the cursor to the last point you digitized to show the intended paths before you digitize the path segments. A dashed outline surrounds the preview paths to indicate the minimum clearance rule. Small arrows and alignment marks appear when the pointer aligns with a nearby path, pin, or via that is on the nets you are routing.

   When starting a path from a pin or an existing path, the routing layer is set to the same layer as the pin or path. When starting a path at a coordinate where there is overlapping

layers or an object with more than one layer a dialog box appears showing you what layers are active and asks you to choose one.

4. Finish the path using one of the following methods.

   - From the Layout pop-up menu, choose *Finish Routing*.

     If the routing search area is not obstructed, the connection will be finished from the last digitized point. An area for the path must lie within the routing search area or the automatic operation is terminated.

   - Continue digitizing points to complete one or more connections.

   - Press the `Return` key.

     The path is ended at the last digitized point.

Changing Layers and Adding Vias

To change layers and add vias while routing, do one of the following:

- From the Layout pop-up menu, choose *Add Via.*
- Press the `space bar`.

The Add Via form appears and all available vias and routing layers that can be reached from the current routing layer are displayed. Choose the layer you want to switch to or choose a via type on the layer.

1. When routing multiple paths, you can change the placement of vias by choosing *Via Pattern* from the middle button Layout pop-up menu.

   You can choose from several predefined via patterns: *Perpendicular*, *Diagonal 1*, *Diagonal 2*, *Stagger*, *Out Taper*, and *In Taper*.

2. To display legal vias sites, turn on *Via Assistance* in the Route Options form.

   When *Snap* or *Display* are turned on legal via sites will be displayed as concentric circles. Single circles are via sites that are not reachable from that location.

Routing Power Rings

You might need redundant wires on selected nets for increased current capacity or to create power rings.

1. Create an ASCII `.do` file with the following syntax:

   ```
   rule net netName (allow_redundant_wiring)
   ```

2. Choose *Route – Options*.
3. In the *Do File* text field, type the name of the `.do` file you created in step one.
4. Click *Refresh*.
5. From the Route Options form, turn A*llow Redundant Wiring On Enabled Nets.*

   Redundant wires will be allowed on nets with the allow redundant wiring rule.

Editing Routes

You can use the following Virtuoso XL wiring editing commands to edit the routed paths you have created.

Command	Description
Edit – Stretch	Pushes routed paths and components while stretching paths
Edit – Other – Split	Reshapes wires by stretching a selected segment and pushes routed paths and components
Edit – Copy Route	Copies existing routes, including vias, to unrouted connections that have similar lengths and topology
Edit – Critic Wire	Eliminates notches and removes extra bends in selected paths.
Edit – Pull	Compacts routes within a selected area
Verify – Check Route	Checks the current cellview for violations based on the options selected in the *Check Routes* form

Verifying routes

Conflicts and design rule violations are prevented using *Interactive Checking*. Some types of rules are not checked. Rules such as same net, and corner to corner spacing rules are not checked. Check specific types of violations by doing the following:

1. Choose *Verify – Check Routes*.
2. Turn on the options you want to check and click *OK*.
3. Select *Verify – Markers – Find* to view violations.

Creating Reports

Use the *Verify – Reports* command to create a temporary file which is displayed in a separate window. To create reports, follow these steps:

1. Choose *Verify – Report*.
2. Select the type of report you want to create from the tabs on the Report form.

The following report options are available:

- *Routing status*

  The status report will vary according to the options you have available. The routing summary includes information about the completion rate, unconnects, wiring, and the conflicts and rule violations encountered in each routing pass. Design statistics include the number of pins, vias, t-junctions, and conflicts, and length information, for each routing layer.

- *Network information*

  Creates a report about all the nets in the design. The Network report includes information about the net names, number of component pins on each nets, number of vias used to route each net, number of junctions used in the net, the length of routed nets.

- *Components*

  The component report lists placement data for the specified component, and position and net data for each component pin.

- *Nets*

  The net report provides a detailed description of a net, including name, network data, connectivity, routing data, and current rules that apply to the net.

- *Rules*

  The rules report contains information about current design rules and constraints. Clearance rules are listed separately for each object to object setting.

Export the Design

1. Choose *Route – Export* To Router.
2. In the Export to Router form turn on the *Use Rules File* option and choose *Set File*. Choose the rules file and click *OK*.
3. It is recommend that a directory be created to store the Virtuoso custom router working files by entering a directory name in the *Export to Directory* field. Otherwise all the routing files will be generated in the default work area.
4. Set the *Default Pin Connection* to *Weak Connection*. The router will connect to only one of the identical pins on each net, preventing routing through (*poly*) gates. If they are already connected, the router uses a pair of weakly connected identical pins as a feedthrough.
5. Choose the *Export marker* option to export markers to the Virtuoso custom router.
6. Scroll down to the bottom of the Export to Router form and choose *Save Defaults*. You can save the form information to an ASCII file.
7. Choose the *Start Router* option and click *OK* to translate the design and turn on the router. The router can also be turned on by typing *vcr* in an xterm from the Export Directory specified during the translation phase. If you turn on *vcr* from the xterm window choose the <cell_name>.dsn file to open the placement data. The .dsn (i.e., design) file is created during the Export step.

   Note: It is recommended that the Export to Router form be used to start the Virtuoso custom router from within Virtuoso XL. The MPS (Message Passing System) is activated and routes in the Virtuoso custom router are automatically transferred back to the Virtuoso XL layout view once the session file is saved (last step in the route process). The Virtuoso XL placement updates and parameter changes are also reflected in the Virtuoso custom router with MPS active. See the "Message Passing System" on page 224 section for more details.
8. View the translation messages in the CIW or xterm window from where layoutPlus was turned on. A message indicating successful translation of the design to the router should appear.

   Note: If the pcell has been coded with a bulk contact defined on nwell/pwell wellbody, warning messages will appear during translation. These warning messages can be ignored. Avoided these warning messages by translating nwell/pwell well body as routing layers (using Layer Add in the translation form), and then by turning off routing directions for these layers. This workaround can cause accidental routing using these layers.

Using the Router

The placement exported to the router should already have devices and cells placed either automatically using the Virtuoso custom placer or in a manual assisted mode using Virtuoso XL in a way that facilitates routing and has sufficient room for the interconnects. Minor placement changes can be made in the Virtuoso custom router using the interactive placement options.

1. Choose the *Place* icon to switch to placement mode.
2. Choose *Select – Components – Mode* to select the devices. Choose the *Move Component* icon to move the selected devices and the *Push Component* icon to push the devices while shoving other devices out of the way.
3. Components can be swapped, if needed, by selecting the *Trade Components* icon (for example to shorten a net). Align and space the components by selecting *Align Components* and *Space Components* icons respectively.
4. Choose *Define – Component Cluster – Define/Forget* by List to cluster abutted devices.

   Note: Any kind of placement change in the Virtuoso custom router environment is reflected back in to Virtuoso XL through the Message Passing System. MPPS can be slow at times. It is recommended to make any major placement change in Virtuoso XL. Specifically abutment and unabutment should not be done in the Virtuoso custom router.
5. The pins are also exported to the router from Virtuoso XL. When creating pins in Virtuoso XL the layers the pins are placed on will need to be available in the cell for routing. For example if pins are placed on metal3 but routing is restricted in the cell to only *metal1* or *metal2* the router will not be able to complete the connections.
6. Net-based constraints can be added through the Virtuoso constraint manager and translated to the router. Constraints entered using the Virtuoso constraint manager are extracted and placed in a <cell_name>Constraints.do file in the working directory during translation. The `.do` file is automatically located by the Virtuoso custom router during invocation. Net-based constraints are not supported in Virtuoso XL. Power, ground and critical nets should be identified using the constraint manager (see the “Constraint Manager” on page 200 for more info).
7. Constraints can also be added within the router environment by creating rules pertaining to clearance, timing, IR drop, crosstalk and shielding for the various Nets/ Net Classes/ Net Layers etc. Constraints can be created from the Rules menu. These rules/ constraints can also be put in a `.do` file and executed at the prompt.
8. The router is a full-featured environment for both interactive single net routing and automatic multi-net routing. The recommended methodology is to first route power, then the critical nets and finally the remaining nets. Routing the remaining nets should be

focused on using metal 1 for most routes, using *metal2* and *poly* once *metal1* options have been exhausted (making sure routing direction is maintained for a layer if feedthroughs are required in the cell in that layer).

Message Passing System

The message passing system (MPS) is the system of communication between DFII and the router. MPS enables dynamic instance updates from Virtuoso XL to the router.

1. In the DFII layout window choose *Connectivity – XL Probe*. Choose a net or device to be probed. The net or device is highlighted in the schematic, layout, and the router window.
2. Choose *Edit – Move* to move an instance in the layout window. The router window is automatically updated.

   Note: If the placement change in Virtuoso XL is a violation in the Virtuoso custom router (for example, placing outside the prBoundary, placing components so they overlap etc.), the router and MPS will not allow the change to be made.
3. To change the component parameters in Virtuoso XL and have the changes updated in the Virtuoso custom router, choose the instance and then choose *Edit – Properties*.
4. You can send commands to the router from DFII using SKILL routines.
5. Save a session file in the router and automatically update the layout cellview.

   Note: MPS can be slow at times so it is recommended that any major placement changes be made in Virtuoso XL. Abutment and unabutment should not be done in the Virtuoso custom router as this behavior is not totally predictable through MPS. If devices need to be regenerated using the *Design – Gen from Source* command in Virtuoso XL, you should quit out of the router session.

Example of a Translations Rules File

Below is a sample of a translation rules file.

```
iccLayers =

list(
list( '("M3" "drawing") "metal" "vertical" 0.28 0.28 nil t)
list( '("V2" "drawing") "cut" "off" 0.26 0.26 t t)
list( '("M2" "drawing") "metal" "horizontal" 0.28 0.28 nil t)
list( '("V1" "drawing") "cut" "off" 0.26 0.26 t t)
list( '("M1" "drawing") "metal" "orthogonal" 0.23 0.23 nil t)
list( '("CO" "drawing") "cut" "off" 0.22 0.25 t t)
list( '("PO" "drawing") "polysilicon" "orthogonal" 0.18 0.25 nil t)
```

```
iccVias =

    list(
      list( '("tsmc18" "M1_POLY1" "symbolic") t)
      list( '("tsmc18" "M2_M1" "symbolic") t)
      list( '("tsmc18" "M3_M2" "symbolic") t)
    )


iccEquivalentLayers =

    list(
      list(
          '("M3" "drawing")
          '("M3" "net")
          '("M3" "pin")
      )
      list(
          '("M2" "drawing")
          '("M2" "net")
          '("M2" "pin")
      )
      list(
          '("M1" "drawing")
          '("M1" "net")
          '("M1" "pin")
      )
    )


iccBoundaryLayers =

    list(
      list( '("M3" "drawing") '("prBoundary" "drawing") 0.0)
      list( '("V2" "drawing") '("prBoundary" "drawing") 0.0)
      list( '("M2" "drawing") '("prBoundary" "drawing") 0.0)
      list( '("V1" "drawing") '("prBoundary" "drawing") 0.0)
      list( '("M1" "drawing") '("prBoundary" "drawing") 0.0)
      list( '("CO" "drawing") '("prBoundary" "drawing") 0.0)
      list( '("PO" "drawing") '("prBoundary" "drawing") 0.0)
)


iccScopes =
list(
)
iccKeepouts =
list(
    list( nil list(
      list(
          "c-not" list('("NP" "drawing") '("PO" "drawing"))
          '("PO" "drawing") "routing" t
      )
      list(
          "c-not" list('("PP" "drawing") '("PO" "drawing"))
          '("PO" "drawing") "routing" t
```

```
        )
        list(
            "=>" list('("NP" "drawing"))
            '("PO" "drawing") "routing" t
        )
        list(
            "=>" list('("PP" "drawing"))
            '("PO" "drawing") "routing" t
        )
    )   32)
)

iccConductors =
list(
)
```

Power Routing Technical Details

The router is a full-featured environment for both interactive single net routing and automatic multi-net routing. The recommended methodology is to first route power, then the critical nets and finally the remaining nets. Routing the remaining nets should be focused on using *metal1* for most routes, using *metal2* and *poly* once *metal1* options have been exhausted (making sure routing direction is maintained for a layer if feedthroughs are required in the cell in that layer).

Follow the steps below for routing power (An "Example Proute.do file" has been provided).

Power Routing using the Power Router

1. Choose *Autoroute – Power Route – Power Router*.
2. In the Power Route form turn on the *From Pin To Trunk* option if the power rails run on top or below rows of devices. If the rails are in the form of a grid running across the devices (or slotted buses, for example) use the *Via on Mesh* option.

   Note: For pin to trunk routing, choose the *From Pin to Trunk* option. The *Protect Trunks* option can be used to protect the trunks if desired. For routing grids, choose *Via on Mesh*.The *Via on Mesh* option requires you to choose a source layer (i.e., the one corresponding to the rails) and one or more target layers for the vias to be dropped.
3. Check *Specify Nets* and choose the power nets (for example, `vcc` and `gnd`).
4. Check *Via Array Count* and choose the appropriate number of rows and columns for the via array (based on the width of the power rails and electrical/ IR drop constraints). If no value is specified the default behavior is to extend the via array across the entire width of the rails.
5. Choose *All Components* and click *OK*.
6. The power router completes routing based on the above and reports unroutes/ failures if any.

Power Routing using the Standard Router

1. To place the power and ground rails in a special net class choose *Define – Class – Define / Forget by Net*. Choose *Create Class* and provide the class name(i.e., power). Choose the power and ground nets that belong to this class (i.e., vcc and gnd).
2. The proper via size (and type) will also have to be determined to cover the width of the power rail. Use the following syntax either in a `.do` file or type directly at the prompt:

```
circuit class power (use_via (use_array M2_M1 2 1))
```

This tells the router to use a 2x1 M2_M1 via array for nets belonging to class ‘power’.

3. The following syntax can be used if the exact size or kind of via array that might be required is not known. It can be used to determine what is needed to cover the entire width of the power rails if they have different widths:

   ```
   circuit class power (use_via (use_array M2_M1))
   rule class power (extend_via_array on)
   ```

   Note: The *Via Array* option is not available from the pull down menu.

4. If power routing is to be restricted to only using certain layers (i.e., *metal1*), turn off all other layers in the Layers form by selecting the *Layers* icon.

5. Use the *Select – Nets – By Class* to complete the routing by selecting nets belonging to class power. Autoroute with appropriate number of passes.

   Note: If router can not complete the power routing using the chosen layers within a reasonable number of passes, enable the next higher metal layer and try more routing passes.

6. Choose *Edit – Protect – Wires By Class* when routing is complete to protect the power nets. Deselect nets and proceed with the rest of the routing.

   Note: The recommended methodology is that the power and ground rails already exist and are defined in the design prior to exporting to the router. If the power and ground rails do not exit rails will need to be created in the Virtuoso custom router. Use the *Select – Wiring Polygons – Mode* command and switch to the appropriate routing layer. Use the *Edit Route* command to create the polygons for the power rails.

7. If the placement was generated by the Virtuoso custom placer with the option for creating power rails, the power nets show up as trunks. If the power router *Pin to Trunk* option is not being used then the commands undefine or forget the power nets as trunks should be run. Choose the nets and click on *Define-Wires/Pins As Trunk*. Click *Selected Wires* and *Forget*, and click *OK*. Deselect the wires by using the *Select – Unselect All Routing Objects* command.

8. Power nets (or any other nets) can be defined as trunks at any point. Route these nets using the regular routing command to get the pin to trunk type of routing where the subnets are tapped off from the main trunks.

9. Choose the nets and click *Define – Wire/Pins As Trunk*. Choose *Selected Wires and Define* and click *OK*.

10. Route the selected wires for the desired number of passes (choose enable or disable on certain layers to constrain the routing).

Note: The equivalent command line syntax would be:

```
sel net <net name>
assign_supply * (selected_wires)
edit_enable_layer <layer name> off
route <#passes>
```

Refer to the *IC Shape- Based Technology Chip Assembly User Guide* for more information on power routing.

Example Proute.do file

```
# power routing do file using PROUTE PIN_TRUNK
# REV: 0.1
#------- SELECT ROUTING LAYERS ---------------
#Turn on METAL1
sel layer METAL1
edit_enable_layer METAL1 on
direction METAL1 vertical
#Turn on METAL2
sel layer METAL2
edit_enable_layer METAL2 on
direction METAL2 horizontal
#Turn off METAL3
unsel layer METAL3
edit_enable_layer METAL3 off
#Turn off POLY
unsel layer POLY1
edit_enable_layer POLY1 off
#---------- SET VIA RULES ----------------------
#all pin_trunk to route when component pin overlaps trunk
rule ic (stack_via any_overlap)
rule ic (clearance 0 (type via_via_same_net))
#---------- ASSIGN POWER NETS AS TRUNKS ----------
#assign pre-routed power and ground as trunks (supply)
sel net vcc
assign_supply vcc (selected)
unsel net vcc
sel net gnd
assign_supply gnd (selected)
unsel net gnd
#--------- SELECT COMPONENETS AND RUN the POWER ROUTER -
#select componenets for proute to route from pins to trunks
sel all components
#run the power router
#set min_trunk_width so pins are not selected as trunks
proute pin_trunk (net vcc) (net gnd) (min_trunk_width  0.8)
unsel all comp
unsel all net
sel net gnd
sel net vcc
route 5
#
```

```
#----- CLEANUP THE ROUTER ENVIRONMENT ------------
#cleanup
sel net vcc
unassign_supply vcc (selected)
unsel net vcc
sel net gnd
unassign_supply gnd (selected)
unsel all net
unsel all components
write wires ./vcr/data/power.w
#----- End Proute --------
```

Another Example: Power.do file

```
#write wires ./vcr/data/preroute.w
define (class power)
define (class power (add_net  gnd))
define (class power (add_net  vcc))
unselect layer METAL3
edit_enable_layer METAL3 off
unselect layer POLY1
edit_enable_layer POLY1 off
unselect layer METAL2
edit_enable_layer METAL2 off
change escape_distance 2
circuit class power (use_via (use_array M2_M1))
rule class power (extend_via_array on)
sel class power
route 25
edit_enable_layer METAL2
route 25 16
#protect class  power
#write wires ./vcr/data/power.w
unselect all nets
```

Routing Steps Technical Details

Critical Nets Routing

Once power routing is completed route the critical or higher priority nets. You will create a class to put the critical nets in.

1. Choose *DefineClass – Define/Forget By List*. Choose *Create Class* and change the name to `critical`. Choose the critical nets from the list to add to the class. Use the *Select – Nets-By Class* command to choose the critical class. Choose *Autoroute – Detail Route – Detail Router* and specify the appropriate number of passes.
2. If there are several priorities of critical nets then a separate class can be created to identify each priority of a net and then have a sequential list of route commands corresponding to the list of priorities.
3. Critical nets can also have additional constraints specified either individually or by class. For example, some of the critical nets might have special requirements for width and spacing (between net pairs) or may need to be shielded. Some other nets may need to be routed as differential pairs.
4. Constraints can be set through the Virtuoso constraint manager. See the The Virtuoso Constraint Manager on page 151 section for more details as to how to set routing constraints.
5. There are several ways of setting constraints on nets in the Virtuoso custom router. Choose the *Rules – Net* menu in the Virtuoso custom router to set constraints that pertain to clearance, width, shielding, timing, crosstalk etc. Use the *Rules – Class* and *Rules – Class To Class* commands Options to specify constraints for net classes and between net classes.

   Note: Even though nets are typically shielded inside the Virtuoso custom router, it is also possible to shield critical nets in Virtuoso XL by using the *Create – Multipart Path* command. Refer to the placement section for details.
6. An example file ("The criticalnet.do file") is included below to show how to control critical nets by using a `.do` file. Running this do file will automatically route the critical nets defined in the file.

Critical Nets chapter for more information on critical nets routing. The Virtuoso custom router is a constraint driven router that is driven by routing rules. For more information refer to Chapter 3, "Setting Routing Rules" chapter of the *IC Shape-Based Technology Chip Assembly User Guide*.

Regular Nets Automated Routing

1. Once the power and critical nets have been routed, the rest of the nets can be routed with an emphasis on minimizing routes on specified layers. For example poly routing needs to be minimized for performance reasons. This requires that *metal1* and *metal2* be used first if possible and that *poly* be used last and with a higher cost in the routing scheme. An example `.do` file "Route.do file" is provided.

   Note: The direction of routing cases where feedthroughs are required will need to be controlled. Over-the-cell routing, if permitted, will need to allow feedthroughs in the cells. When the cell is routed at the device-level the metal layers will need to be restricted so that some metal layers will be left for routing at the next level of hierarchy. For example if vertical *metal2* feedthroughs are required then *metal2* will need to be vertical in the cell to reduce the number of tracks that are taken up by a route. This is because a vertical *metal2* line, in this case, will take up only one track while a horizontal track will take up multiple tracks.

For more information refer to Chapter 3, "Setting Routing Rules" of the *IC Shape-Based Technology Chip Assembly User Guide*.

Example Do files

The criticalnet.do file

```
#########################################################
#   criticalnet.do -- VCR Router
#   To define a new class called 'critical' (or what
#   name you choose), uncomment the following lines
#   and then'add_net' the net names you want.
#   Execute this file in Vcr prior to routing all other nets


define (class critical)
define (class critical (add_net  ph1_c))
define (class critical (add_net  ph1b_c))
define (class critical (add_net  ph2b_c))
# The above lines can be omitted if the class and nets are defined in
Virtuoso XL.
# The class will be defined automatically during the translation step

sel class critical
rule class critical (width .35)
sel class critical
route 5
protect class critical
```

```
write wires critical.w
unselect all nets
#########################################################
```

Route.do file

```
#########################################################
#   Route.do -- VCR Router
#  Set Met1 to be the 'primary layer'.  The direction
#  of the layer is set in the translation rules, but
#  if want to override in the command line, can do
#  as did below - otherwise change it manually on the
#  layers panel.
#
#direction met1 vertical
select layer met1
edit_active_layer met1
#####
#  Set a high tax on the useage of via/contacts,
#  and set the escape distance.
#
tax via 7
#change escape_distance 2
#
#####
#  If want to enforce the direction of the layers per
#  the layers panel (example: all met1 tries to
#  maintain horiz, met2 = vert, etc) then uncomment
#  the 'limit way' command below.
#
#limit way 5
#
#####
#  Start off by setting all routing lyrs other than
#  met1 to be off - and to have a med/high cost to
#  use them.
unselect layer met3
edit_enable_layer met3 off
cost layer met3 high ( type length )
#
unselect layer met2
edit_enable_layer met2 off
cost layer met2 medium ( type length )
#
```

```
unselect layer poly
edit_enable_layer poly off
cost layer poly high ( type length )
#
#####
#  Initial autoroute passes (25) with only met1
#  available to route, trying to force as much
#  routing on met1.  Clean passes will eliminate
#  unnecessary bends and unnecessary vias on wires.
route 25
clean 2
#
#####
#  Now turn on met2, and allow some met2 connections
#  if needed.  Having the router start at pass# 16
#  in the internal algorythm, and execute 25 passes,
#  unless if it completes sooner.
#direction met2 horizontal
select layer met2
edit_enable_layer met2 on
rule layer met2 (max_segment 3)
rule layer met2 (limit_way 1)
route 25 16
clean 4
#
#####
#  Now turn on poly, and allow some poly connections
#  if needed.
#direction poly vertical
select layer poly
edit_enable_layer poly on
rule layer poly (max_segment 3)
route 25 16
clean 4
#
#####
#  Write the route status out to a file.  Shows all
#  autoroute progress, and layer utilization, etc.
#
report status routestatus.rpt
#####
#  Write out the route and place conflicts.
#
report conflict conflict_place.rpt (type place)
report conflict conflict_route.rpt (type route)
```

```
#####
#  Write out the 'session' file to send back to Virtuoso XL.
write session adc_comparator_actr.ses
#
##########################################################
```

Routing Analysis Technical Details

Interactive Routing

It may not be possible to achieve 100% route completion on the first pass. The layout t may need to be further optimized. Reasons for less than 100% completion or non-optimal results include router failure to find a path due to poor placement, high congestion, a blocked pin, and/or over constrained nets.

1. To overcome poor placement or problems due to high congestion, manually route the net in the Virtuoso custom router. The Virtuoso custom router interactive route capability can be used to thread the route through the congested area. Use the Virtuoso custom router to automatically complete the route once the congestion is manually cleared.
2. To perform interactive editing use the *Select – Wires – Mode*. Choose the *Edit Route* icon and change to the correct routing layer. At any point during routing you can change the wire width or add a via added by clicking the right mouse button and selecting *Use Width* or *Add Via*.
3. If the entire route is completed interactively you are still recommended to run a couple of cleaning passes by using the *Autoroute – Clean* command. Use the *Specify* option or number of passes. The *Clean* command will cleanup any extraneous jogs that may have been added along the way.
4. If interactively routing is not possible, routes can be ripped up, either partially or completely. Use the *Select – Wires – Mode* command and then choose *Edit – Delete Wires – Selected*. Some placement adjustment can also be done by changing to *Place* mode. The router can then be used to make some additional passes (route 25 16; clean 2) in order to route all the nets.
5. If there is a need to significantly adjust the placement and a need to re-route the entire cell, all routes and settings can be cleared after unprotecting the relevant nets. Choose *Edit – Unprotect – Wires By Net* and *Edit – Delete Wires – All*.
6. If you are interactively routing to the pin, but are prevented from attaching the pin to the net, then the pin is blocked. The cause of a blocked pin is that there is no DRC correct way to get to the pin because of placement. Translation rules may also cause this issue, (i.e., there might be a path to the pin in layout but the keepout regions generated during translation could have blocked access to the pin). There is a pin cut to edge capability in the translation rules which can be used.

   Note: If the design was imported to the Virtuoso custom router with substrate/ well taps, guard rings etc., they may show up as unconnects in the Virtuoso custom router. These shapes are created with layers, such as wells, implants etc., that are not translated as

routing layers but as reference layers. The Virtuoso custom router flags them as incomplete nets because the associated layers are reference layers with implied connectivity, but would not be able to hook them up because these are not on actual routing layers.When the design is imported back to Virtuoso XL, these do not show up as opens because Virtuoso XL understands the connectivity for the layers that have been defined in the technology file.

7. Correcting pin placement is done in Virtuoso XL. The pin locations are locked in the Virtuoso custom router. The routing is transferred back to the Virtuoso XL layout window by invoking the *Route – Import from Router* command. If there is no reason to keep the routes obtained from the Virtuoso custom router they can be discarded by using the *Design – Discard Edits* command. In Virtuoso XL the pins can be reposition to allow the route to complete successfully. This can involve moving the pins or other design elements such as the prBoundary to allow more room.

8. If there are constraints placed on a net which prevent the net from being completely routed (i.e if the net is overconstrained) then the Virtuoso custom router may fail. The Virtuoso custom router will relax the constraints and report this in the routing results file. Choose *Report – Route Status* to view he routing results. Choose *Report – Rules* and click the *Specify* option to view any specific rule violation.

9. Route completion is performed in the Virtuoso custom router. The router is able to enforce and maintain design rule correct layout during both placement and routing modifications. The connectivity consistency with the schematic will be maintained in both the Virtuoso custom router and Virtuoso XL.

10. Once the routing results are satisfactory, the design can be imported the back to Virtuoso XL by selecting the *Route – Import from Router* command. The routing information will automatically be imported if the session file is saved.

For more information on the Virtuoso custom router refer to the *IC Shape-Based Placer/ Router Documentation*.

Post Route Steps in the Virtuoso XL Layout Editor

1. After importing the design back to Virtuoso XL, run a first pass check of the connectivity by selecting *Connectivity – Check-Shorts and Opens.*

2. If there are connectivity issues with any net, the entire net can be traced by selecting *Connectivity – Mark Net*. Click on a net, and the entire net is highlighted.

   Note: The viaLayers and lxNoOverlapLayers must have been defined in the technology file in order to use the *Mark Net* command.

3. Shorts/ opens if any, should be investigated. Taps and guard rings should be connected at this point to the appropriate supply sources (if they were not connected already prior to sending the design to the router) These connections are not supposed to be handled by the Virtuoso custom router. Refer to the Post Placement Steps in the Virtuoso XL Layout Editor for more information regarding tap insertion and connection.
4. Run DRC and LVS on the completed layout by selecting *Verify – DRC*, *Verify – Extract* and *Verify – LVS*. Refer to "Physical Verification" on page 154 for more details.

Constraints File Syntax

The constraints manager reads and writes constraints using a design constraints description language syntax.

The syntax for the constraints file consists of command keywords followed by arguments that are preceded with a hyphen (-). Each argument includes appropriate values. Values are specific to the argument and must be of the type required by the specific argument. For example, some arguments need a string value and some need a number. Each entry is separated with white space (tab, blank, or eol). Each command has a specific number of arguments and all must be present. They can be entered in any order and are not case sensitive. New commands should begin on a new line but this is not required. The pound sign (#) indicates that the rest of the line until the eol character is a comment.

Comments are not saved in the constraints database so if a file with comments is loaded and then dumped the file back out the comments would be lost. While commands and options are not case sensitive the values should be considered to be case sensitive and should match value in the database.

Design Constraints Language Syntax

The table below lists data types used in the following syntax descriptions.

Data Types

Prefix	Internal Name	Data Type
f	flonum	floating-point number
l	list	linked list
t	string	character string (text)
x	integer	integer number

The syntax of the constraint manager file is defined as:

CURRENT_SCOPE
The CURRENT_SCOPE line describes the library and cellview information. The line must be present before any constraint syntax.

Syntax
```
CURRENT_SCOPE -lib t_libName -cell t_cellName -view t_viewName
```

Example `CURRENT_SCOPE -lib overview -cell follower -view layout`

NET_CLASS

The NET_CLASS line must be present before any constraint line that references the net-class.

Syntax `NET_CLASS -name t_constraintName -netlist {l_netNames}`

Example `NET_CLASS -name cmxNC3 -netlist {net20,net22,net10}`

NET_NCLASS

Syntax `NET_NCLASS -name t_constraintName - x_constraintWeight -priority x_priorityValue -width f_widthValue -minird f_minIrValue -maxird f_mxIrValue -netlist {netName}/nil -nclist {netClassName}/nil`

`Only one of the string values is allowed. The other must be nil.`

Example `NET_NCLASS -name cmxNetCatCon_26 -weight 200 -priority 20 -width 2.000000 -minird 0.000000 -maxird 0.000000 -netlist {Vcc} -nclist nil`

NET_NET

Syntax `NET_NET -name constraintName -weight x_constraintWeight -clearance f_clearValue -diffgap f_diffGapValue -paragap f_paraGapValue -parawidth f_paraWidthValue -depth x_depthValue -overhang f_overhangValue -netlist {l_netNames}`

Example `NET_NET -name cmxNetToNetCatCon_22 -weight 200 -clearance 2.500000 -diffgap 0.000000 -paragap 0.000000 -parawidth 0.000000 -depth 2 -overhang 0.000000 -netlist {NMbias,Out}`

NETCLASS_NETCLASS

Syntax `NETCLASS_NETCLASS -name constraintName -weight x_constraintWeight -clearance f_clearValue -depth x_depthValue -nclist {l_netClassNames}`

Example `NETCLASS_NETCLASS -name cmxClassToClassCatCon_29 -weight 200 -clearance 3.000000 -depth 2 -nclist {cmxNC4,cmxNC3}`

CLASS_NET

Syntax
```
CLASS_NET -name t_constraintName -weight x_constraintWeight -clearance f_clearValue -depth x_depthValue -nclist {t_netClassName} -netlist {t_netName}
```

Example
```
CLASS_NET -name cmxClassToNetCatCon_30 -weight 200 -clearance 2.000000 -depth 2  -nclist {cmxNC3} -netlist {net22}
```

AXIS

The AXIS support entity line must be present before any constraint line that references the axis.

Syntax
```
AXIS -name t_constraintName -dir NS/EW -xcoord f_xCoordValue -ycoord f_yCoordValue
```

Example
```
AXIS -name cmxAxis0 -dir NS -xcoord 0.400000 -ycoord 0.000000
```

CLUSTER

The CLUSTER support entity line must be present before any constraint line that references the cluster.

Syntax
```
CLUSTER -name t_constraintName -instlist {l_instNames} /nil -pinlist {l_pinNames} /nil -shapelist {l_shapeNames} /nil -clusterlist {l_clusterNames} /nil
```

Example
```
CLUSTER -name cmxClstr_0 -instlist {|Q8} -pinlist {Pbias|Pbias,In|In,NMbias|NMbias,Nbias|Nbias,gnd!|gnd!,Out|Out} -shapelist nil -clusterlist nil
```

SYMMETRY

Syntax
```
SYMMETRY -name t_constraintName -weight x_weightValue -axis t_axisName -instlist {l_instNames} /nil -pinlist {l_pinNames} /nil -shapelist {l_shapeNames} /nil -clusterlist {l_clusterNames} /nil
```

Only one of the list values is allowed. All others must be nil.

Example
```
SYMMETRY -name cmxSymCatCon_31 -weight 200 -axis System_Horizontal -instlist {|Q9,|Q10} -pinlist nil -shapelist nil -clusterList nil
```

FIXED

Syntax

```
FIXED -name t_constraintName -weight x_weightValue -checkx
TRUE/FALSE -checky TRUE/FALSE -coord
f_coordValue:f_coordValue -orient AS_IS/NO_CARE/R0/R90/R180/
R270/MX/MY/MXR90/MYR90 -refHandle CENTER/LOWER_LEFT/UPPER_RIGHT -
inst t_instName /nil -pin t_pinName /nil -shape t_shapeName /
nil
```

Example

```
FIXED -name cmxFixedCatCon_32 -weight 200 -checkx TRUE -checky
TRUE -coord 70.400000:50.200000 -orient NO_CARE -refHandle CENTER
-uMove FALSE -inst nil -pin Out|Out -shape nil
```

GROUP

Syntax

```
GROUP -name t_constraintName -weight x_weightValue -
preserverelative TRUE/FALSE -noFence TRUE/FALSE -fence
t_fenseName -exclInsts t_instNames /nil -exclPins t_pinNames /
nil -exclShapes t_shapeNames/nil -exclAll TRUE/FALSE -instlist
{l_instNames} /nil -pinlist {l_pinNames} /nil -shapelist
{l_shapeNames} /nil
```

Example

```
GROUP -name cmxGroupingCatCon_23 -weight 200 -preserverelative
TRUE -noFence TRUE -fence nil -exclInsts nil -exclPins nil -
exclShapes nil -exclAll FALSE -instlist {|R4,|R5,|R11,|R12} -
pinlist nil -shapelist {rect9}
```

ALIGN_INST

Syntax

```
ALIGN_INSTS -name t_constraintName -weight x_weightValue -
refOrient {SAME/NO_CARE/SAME_OR_MIRRORED_UPSIDE_DOWN/
SAME_OR_MIRRORED_SIDEWAYS/MIRRORED_UPSIDE_DOWN/
MIRRORED_SIDEWAY} -alignHandle {NO_CARE/RIGHT/LEFT/TOP/BOTTOM/
H_CENTER/V_CENTER/SPECIFIC} -alignSeg x_segNumber -innerEdge
TRUE/FALSE -alignLayer t_layerName -ordered on/off -alignwith
{inst,l_instNames}/{shape,l_shapeNames}/{pin,l_pinNames} -
instlist {l_instNames} /nil -pinlist {l_pinNames} /nil -
shapelist {l_shapeNames} /nil -clusterlist {l_clusterNames} /
nil
```

Example

```
ALIGN_INSTS -name cmxAlignCatCon_28 -weight 200 -refOrient
NO_CARE -alignHandle RIGHT -alignSeg 0 -innerEdge TRUE -alignLayer
bBox -ordered off -alignwith {inst,|NAND3_0|M0} -instlist
{|Q9,|Q10} -pinlist nil -shapelist nil -clusterlist nil
```

DISTANCE

Syntax

```
DISTANCE -name t_constraintName -weight x_weightValue -
checkMaxX TRUE/FALSE -maxX f_maxXDistValue -checkMaxY TRUE/
FALSE -maxY f_maxYDistValue -checkMinX TRUE/FALSE -minX
f_minXDistValue -checkMinY TRUE/FALSE -minY f_minYDistValue -
refX CENTER/TOP/BOTTOM/NO_CARE/NEAREST_EDGE -refY CENTER/TOP/
BOTTOM/NO_CARE/NEAREST_EDGE -instlist {l_instNames} /nil -
pinlist {l_pinNames} /nil -shapelist {l_pinNames} /nil -
clusterlist {l_clusterNames} /nil
```

Example

```
DISTANCE -name cmxDistCatCon_24 -weight 200 -checkMaxX TRUE -maxX
3.000000 -checkMaxY TRUE -maxY 3.000000 -checkMinX TRUE -minX
2.000000 -checkMinY TRUE -minY 2.000000 -refX CENTER -refY CENTER
-instlist {|R4,|R5,|R11,|R12} -pinlist nil -shapelist nil -
clusterlist nil
```

Engineering Change Order Cycle Detailed Instructions

Update the Databases

The Virtuoso Schematic Composer schematic has been updated with additional devices and associated connectivity.

1. Start Virtuoso-XL from Schematic and open the corresponding layout view.
2. Choose *Connectivity – Update Components and Nets* to update the layout to include additional devices and nets, and to delete devices and nets that have been removed from the schematic.
3. The *Update Components and Nets* command brings up the Layout Generation Options form which allows you to update instances, pins and the prBoundary. You can also choose automatic folding and chaining options for the additional instances as well as specify the layer and size for any additional pins.

   Note: Turning on folding and chaining in the Layout Generation Options form places devices on top of each other.
4. It is not recommended to regenerate the prBoundary because the pin constraints pertaining to the boundary edges will be lost. If you regenerate the prBoundary to get an estimate of the bounding box based on the updated schematic, then the prBoundary may be regenerated in a separate layout view and the information can be used to update the current layout view.
5. The *Connectivity – Update Components and Nets* command will generate the additional devices with associated connectivity. Devices that have been deleted from the schematic will appear flashing in the layout. These devices will need to be deleted manually.
6. To update any device parameter that might have changed, choose *Connectivity – Update Layout Parameters*. This command will prompt you to choose the devices that need to be updated.
7. Use the *Window – Zoom-In* command to locate the devices that need to be updated. You can then highlight the devices either in the schematic or in the layout (choose all the devices if it can't be easily determined as to which ones have been updated). Use the `shift` key to choose multiple devices. Press return once selecting is completed.
8. The *Connectivity – Update – Layout Parameters* command will update all the mismatched parameters for the selected instances in the layout, based on the modified parameter values in the schematic.

9. Updating the parameters may cause some overlaps or DRC violations which should be resolved manually.
10. Before modifying the placement and re-routing choose *Connectivity – Check-Against Source* to make sure all the devices have been generated correctly and the parameters match between the layout and the schematic.

Analyze Scope of the Engineering Change Order

An analysis needs to be performed to assess the extent of change in order to determine the implementation strategy. The existing design should be used if it has already been characterized for performance. If the change is significant it may be necessary to start over using the *Design – Gen from Source* command. This might be a better approach if the placement was generated using the automatic placer (Virtuoso custom placer) and if all the constraints and the Virtuoso custom router do files can be re-used easily to re-generate the layout. If the change only involves nets then go straight to the router for Routing Changes.

Placement Changes

1. Place the additional devices optimally in the existing design, following the flight lines. Fold, chain and abut (to share diffusion) the devices as required. Verify that the devices with updated layout parameters are correctly placed. Extend the prBoundary or modify the aspect ratio if absolutely required. (The prBoundary can be regenerated using *Update Components and Nets* command and then turn on *just prBoundary* option in the Gen from Source form, with the appropriate size/aspect ratio.)
2. If the placement was generated using the Virtuoso custom placer verify that there is enough room in the rows or in the given placement region for the additional devices. Otherwise it may be necessary to change/resize the rows/regions. In the Placement Planning form change the *Style* to *Manual User Defined* and highlight and modify the appropriate rows, or redraw or stretch the region for area based placement.
3. You may need to lock down some of the critical devices using the *Edit – Other – Lock Selected* command or add a fixed constraint using the constraint manager prior to running the placer in ECO mode. It may be necessary to add some additional constraints for the new devices to address performance issues.
4. Choose *Place – Placer* and turn on the *ECO Mode* option.
5. The placer will generate a placement with minimal change in the existing placement without disturbing the locked devices.

   Note: Currently, the ECO flow does not preserve logical hierarchy.

6. Certain device placements may need to be hand tweaked to meet constraints. The way the placer works in the ECO (and regular) mode with routed design is to ignore routing. So running it through the Virtuoso custom placer ECO mode is the same as merging or reading in a modified placed view (design file) into an existing Virtuoso custom router wires file.

7. Once the devices have been optimally placed in Virtuoso XL/Virtuoso custom placer, the routing can be modified and additional unrouted devices can be hooked up within Virtuoso XL. Export the new design/placement to the router by using the *Route – Export to Router* command.

   Note: If there are routing related violations flagged by DRC, then it may be a good idea to export the markers to the router as well. Turn on the *Export Markers* option in the Export to Router form. You can fix these violations in the router environment.

Routing Changes

1. In the router autoroute using do files to implement the routing constraints (width, spacing shielding, switch layers etc. for selected nets) could be chosen or manual route could be used to route the relevant nets. If the changes are not extensive routing manually will provide better control.

2. To ensure that routing constraints are maintained and executed by the router, appropriate do files will need to be created. Another way of entering these net-based constraints is through the constraint manager prior to translating the design. These constraints are then translated to the router. Translation generates a `.do` file called *`cellName`*`Constraints.do` that has all the constraints in `.do` file syntax. The .do file is executed automatically when using the autorouter. The constraint manager does not support all net-based constraints. Additional constraints can be written to a `.do` file or created through Virtuoso custom router pull down menus.

3. Any previously routed nets will be protected by default when translated to the router. These nets or net segments may need to be deleted in Virtuoso XL in order to allow the router to correctly complete the design. In the router window use the *Select – Wires* and *Edit – Delete Wires – Selected* commands. An alternative is to unprotect all or some nets in the router by using the *Edit – Unprotect – Wires by Net* command. Be careful not to unprotect power or critical nets because they might be re-routed by the router.

4. Verify that all power and critical nets are protected. Unprotect or delete any net that needs to rerouted based on the ECO. Execute appropriate do files and hand edit the nets wherever required. Complete any nets that the auto router could not complete.

Note: If the ECO involves only routing changes (for example, optimizing critical nets, shielding or increasing width or separation of certain nets etc. based on simulation results), then the router can be started immediately to implement these changes.

5. When most of the changes pertain to routing and are conducted inside the router make minor placement changes inside the router. The Message Passing System (MPS) will update these changes in Virtuoso XL. Any significant placement changes should be done by importing the design to Virtuoso XL making the changes, and then exporting it to the router again. Use the *Route – Export From Router* and *Route – Import From Router* commands.

6. In the router window change the mode from route to place by clicking on the *Place* icon. Use the *Push* and *Shove* commands to make room for the new nets, without disturbing the relative placement/position of the devices. The *Swap* command can be used to swap devices if you need to shorten any nets.

7. While abutment and unabutment of devices do work in the router environment through MPS if the pcells are properly coded. Abutted device chains should not be disturbed. Abutted devices and device chains should be grouped by selecting *Define – Component Cluster* and moved together using the *Select – Component Cluster* and *Move* commands.

8. Once the placement and routing changes have been implemented as per the ECO, physical verification should be run on the design to make sure the design is DRC and LVS correct. Use the *Verify – DRC*, *Verify – Extract* and *Verify – LVS* commands from the *Virtuoso XL* menu.