SYNOPSYS®

CUSTOMER EDUCATION SERVICES

IC Compiler 1 Workshop

Lab Guide

20-I-071-SLG-011 2010.12

Synopsys Customer Education Services
700 East Middlefield Road
Mountain View, California 94043

Workshop Registration: **1-800-793-3448**

www.synopsys.com

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, Columbia,Columbia-CE, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, Direct Silicon Access, Discovery, Encore, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, HSIMplus, HSPICE-Link, iN-Tandem, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport,Library Compiler, Magellan, Mars, Mars-Rail, Milkyway, ModelSource, Module Compiler, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Raphael-NES,Saturn, Scirocco, Scirocco-i, StarRC, Star-SimXT, Taurus, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license. ARM and AMBA are registered trademarks of ARM Limited. Saber is a registered trademark of SabreMark Limited Partnership and is used under license. All other product or company names may be trademarks of their respective owners.

Document Order Number: 20-I-071-SSG-011
IC Compiler 1 Lab Guide

0A

IC Compiler™ GUI

Learning Objectives

This lab has two purposes:

1. To familiarize you with the IC Compiler GUI.
2. To learn how to get help with commands and variables.

You will work with a design that has been previously placed by IC Compiler.

After completing this lab, you should be able to:

- Invoke and exit IC Compiler
- Load a saved design
- Configure “layout window”
- Navigate the layout view
- Select and query layout objects.
- Use *help*, *printvar* and *man* to get help and additional information about commands and variables

Lab Duration:

Instructions

Task 1. Start IC Compiler

1. Log in to the *UNIX* environment with the *user id* and *password* assigned by your instructor.

2. Before invoking IC Compiler, we want to remove a *GUI window configuration* file, if it exists. This file will exist if you have previously invoked IC Compiler in this login account – its purpose is to remember the last GUI window configuration you had before exiting the tool, so that the configuration will look the same the next time you invoke the tool. For this lab we need you to start with a default window configuration. If the following file exists in your home directory, delete it:

```
UNIX% rm ~/.config/Synopsys/icc_shell.conf
```

Note: This step is NOT necessary during regular use of the tool. It is ONLY done here to ensure a consistent lab environment.

3. From your login or home directory, change your current directory to *lab0_gui*, which is the working directory for this lab.

```
UNIX$ cd lab0_gui
```

4. Start IC Compiler from the *UNIX* prompt:

```
UNIX$ icc_shell
```

The *xterm UNIX* prompt becomes `icc_shell>`, the IC Compiler shell command prompt.

5. Have a look in your current directory. You can type `ls -a` in a *UNIX xterm* window, or in the IC Compiler shell type:

```
icc_shell> ls
```

You will see that command and output log files were created (`icc_shell.cmd` and `.log`). The *.cmd* file records all commands, including initialization commands invoked during IC Compiler start-up. The*.log* file records commands and command output after tool start-up.

Note: Log file naming can be defined through variables in the initialization file, `.synopsys_dc.setup`.

6. Start the GUI. This is the “on demand” (as needed) method:

```
icc_shell> start_gui or gui_start
```

After a short wait a window labeled *IC Compiler - MainWindow.1* is opened. This window can display schematics and logical hierarchy browsers, among other things, once a design is loaded.

Note: Instead of invoking the GUI "on-demand", you can start the IC Compiler GUI from the *UNIX* prompt: **`icc_shell -gui`**.

7. Load the *placed* cell from the *risc_chip.mw* MilkyWay design library, as follows:

a. In the *MainWindow* click on the little yellow "open design" icon on the top left, or use the menu command: **File → Open Design …**

b. In the *Open Design* dialog panel, click the yellow folder icon . The *Select Library* dialog box opens. *MilkyWay* libraries are marked by an orange "L" icon . Select the library folder ***risc_chip.mw*** and click **Choose**.

c. The middle of the *Open Design* dialog now shows the stored *CELs*. Since there is only one cell (***placed***) in the list, it should already be selected (highlighted in blue). Click **OK** to open it.

A new window labeled *LayoutWindow.1* opens.

8. Bring the *MainWindow* to the foreground and look at the command transcript near the bottom of the window to answer the following question:

Question 1. What command was executed to open the *placed* cell? (scroll up until you find it)

.....

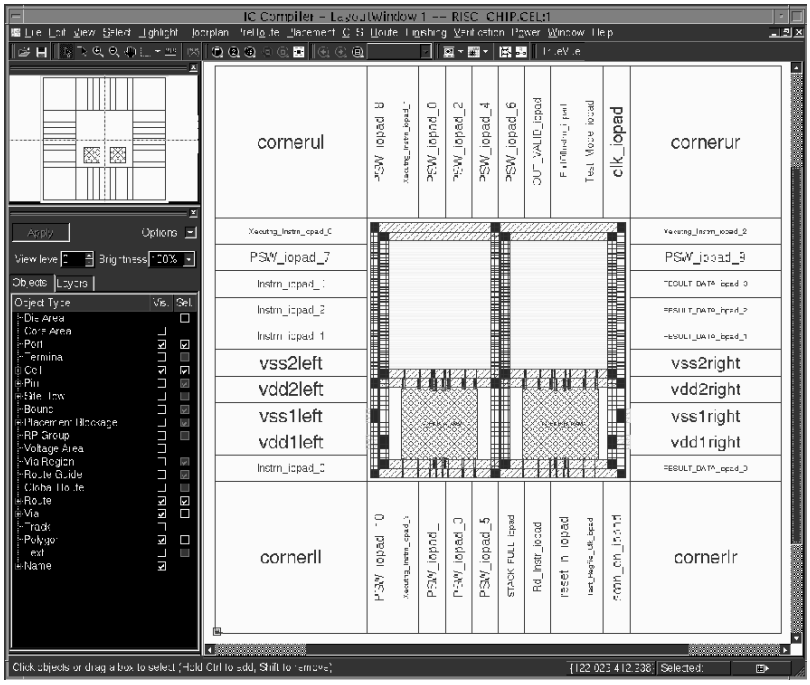
Check your answer against the answer at the end of this lab.

Looking at the transcript is useful to begin to learn IC Compiler's commands. Look at the *UNIX* window where IC Compiler was invoked. Commands can be executed, and are also echoed, there.

9. Bring the *LayoutWindow* to the foreground and enlarge or maximize the window.

10. Press the lower-case **[F]** key to fit the layout to the larger window.

You are looking at the layout view of the design CEL called *placed*, which is part of the *risc_chip.mw* design library. On the outer perimeter of the layout, *IO pad cells* (light blue rectangles) surround the brightly colored center or *core* region on all 4 sides. Between the *core* and the *periphery* or IO pad area there are green and red metal *rings* for power and ground (VDD/VSS). There are also vertical and horizontal VDD/VSS *straps* through the core for better power distribution. At the bottom of the *core* area there are two RAM *macros*. The core and periphery layout, as well as the power routing were defined during the *design planning* phase. During the *placement* phase, the *standard cells* have been automatically placed in horizontal placement *rows* (the darker blue area above the RAM macros). The details of the rows and the standard cells may not be visible. Once you know how to zoom (Task 2) you will better be able to see the standard cells. The blue area is made up of narrow metal lines running horizontally, VDD/VSS *rails*, which distribute power to the standard cells.



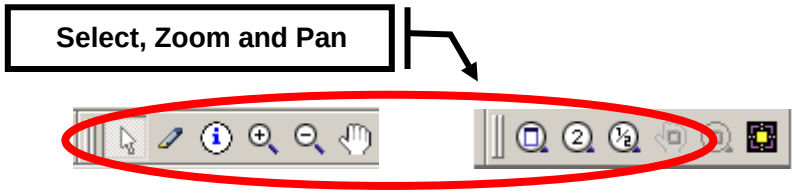
Note: The *LayoutWindow* has its own set of menu entries. Some of these entries are shared with the *MainWindow*, while others are unique to the *LayoutWindow*. Most of the physical processing of the design can be done by commands from this menu system.

Task 2. Navigating the Layout View

1. Spend a few minutes to get familiar with the *zoom* and *pan* buttons in the *LayoutWindow*. While panning and zooming, notice how the yellow rectangle in the *Overview* window (the small "context" window in the upper left corner of the *LayoutWindow*) identifies the area of the design being displayed.

Hint: A short, descriptive 'ToolTip' will pop up when a mouse pointer is held motionless over a button.

To exit the zoom and pan mode pick the 'Selection Tool' (the white arrow icon) or press the [**Esc**] key. The cursor returns to an "arrow" or pointer shape.



Question 2. What is the difference between the "magnifier" button with "2" in it and the button with a "+" in it?

.....

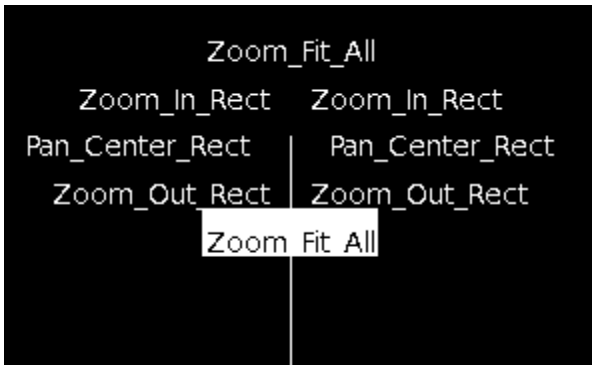
2. "Hot keys" are also available when the *LayoutWindow* is active (i.e. currently selected). Lower-case [**F**] or [**Ctrl F**] both correspond to "zoom fit all" (or full view), for example. [**Z**] is zoom-in.

3. You can find out about other hot key definitions in two ways: Hover with the mouse over a button and a "balloon help" will appear showing the name of the function and the keyboard shortcut. You can also select the pull down menu **Help → Report Hotkey Bindings**. A new view appears, listing the hot key definitions. To close this view select **Window → Close View** or [**Ctrl W**].

4. Some people like to use mouse “strokes” to pan and zoom, instead of using GUI buttons or keyboard “hot keys”. Try using *strokes* as follows:

Zoom in on an area of interest: [**Z**] and [**Esc**].

Now click and hold the middle mouse button while moving the pointer straight up or down and holding it there. The stroke menu appears near the pointer:



Release the middle button and the design should zoom to fit the display window (“Zoom Fit All”). To zoom in on an area “stroke” (move mouse with middle button depressed) in a 45° direction upward (to the left or right) – the view should zoom-in to a rectangular area defined by the stroke. Stroking 45° downward zooms out. Stroking in the east/west direction *pans* the display such that the start point of the stroke is moved to the center of the window.

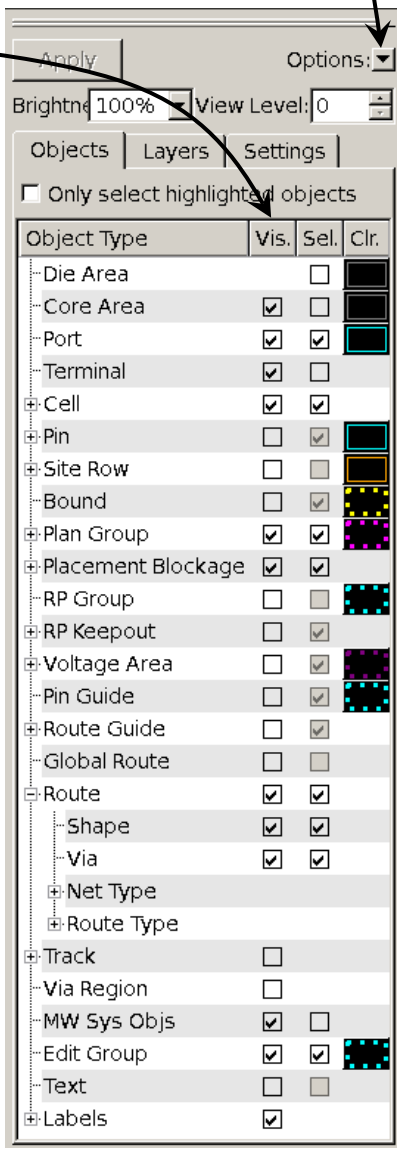
Note: In the interest of time, do not attempt this presently, but it is useful to note that you can query or define your own strokes by using the commands `get_gui_stroke_bindings` and `set_gui_stroke_binding`. “Hot keys” can be defined by using `gui_set_hotkey`.

5. The keyboard **arrow keys** can also be used to pan the display North/South/East/West. Try it.

6. If your mouse has a **scroll wheel**, it can be used to zoom in/out (2X or ½X) around the area of the mouse’s pointer.

Task 3. Controlling Layer Visibility

You can control what types of objects are *visible* and/or *selectable* in the viewing window through the *View Settings* panel. In the following steps you will turn on visibility to some key objects one at a time, to clearly see what they represent. First make sure that under the *Options* pull down menu *Auto apply* is checked: This way selections are applied immediately without having to click on the *Apply* button each time.



a. In the ***Vis.*** column, uncheck everything except ***Cell***. Only the standard-, macro- and IO pad cells are displayed.

b. Now check ***Pin*** as well. The input, output and power connection pins of the cells are displayed.

c. Check ***Route***. All metal *routes* become visible. Since the design has not been routed yet, only power/ground "pre-routes" (from the design planning phase) are seen. You should see power supply *rings* around the core as well as vertical and horizontal *straps* through the core area.

d. Check ***Labels***. Cell and instance names become visible. Expand *Labels* by clicking on the "+" icon on the left. Check ***Pin***. Zoom in [**Z**] on one of the standard cell instances - its pin names are now visible. Fit the view to the window [**F**].

e. Select the **Layers** tab, which can be used to "fine tune" the visibility further on a layer-by-layer basis. At the intersection of the row labeled "METAL(14)" and the column labeled "Shape" click on the **blue square** with diagonal lines. The blue horizontal *METAL* (= metal 1) rails disappear.

f. Make "metal 1" visible again.

g. Select the ***Objects*** tab and re-apply the original visibility settings shown in the panel above.

Question 3. What is the difference between the **Vis.** (visibility) and **Sel.** (selection) columns in the above panel?

.....

.....

7. Select the **Layers** tab and use the colors and fill patterns to answer the following questions:

Question 4. On what layer name and number are the red horizontal power straps?

.....

Question 5. On what layer name and number are the green vertical power straps?

.....

8. You can confirm your findings by hovering the pointer over one of the straps. A "query" window appears, which displays information about the object, including its *layer* name.

Task 4. Selecting and Querying Objects

1. Selecting objects:

To be able to select objects the mouse cursor must be an *arrow*, which denotes "select mode". If your cursor is not in select mode either click the ***arrow*** button  or press the **[Esc]** key.

2. Try selecting different single objects with a left mouse click. A selected object is highlighted in white, and remains highlighted until un-selected, or a different object is selected.

3. Unselect all objects by either clicking on an empty area in the layout, by using the menu **Select → Clear**, or by typing [**Ctrl D**].

4. Select multiple objects in the same area with a left button "drag-and-draw". All objects within the drawn rectangle are selected.

5. Keep what is selected and select additional objects by holding down the [**Ctrl**] key while selecting with the left mouse click.

6. You can cycle through "stacked" objects (multiple objects placed on top of each other) by repeatedly clicking the left mouse button until the desired object is highlighted. Try this by clicking on the corner intersection between the red horizontal, and green vertical power/ground rings.

7. Zoom into the blue core area. Select a handful of standard cells by dragging a selection box around them.

8. Incase it is difficult to notice the highlighted (selected) objects among other bright objects, it is possible to reduce the "brightness" of the unselected objects, thereby increasing the contrast. A "***Brightness***" control is located at the top of the *View Settings* panel.

Reduce the *brightness* to **50%** to see the improved contrast.

9. Querying objects:

By default, when the cursor arrow hovers over an object, the object is lightly highlighted, and a query "summary" window appears in the bottom left, displaying some key attributes of the object.

To obtain a "full query", select a single standard cell, and query it by typing lower-case [**Q**] or by using the menu entry: **Select → Query Selection**. A window opens and lists all the attribute values of the selected cell.

10. Close the query window by clicking the "Hide" **minus sign** in its upper right corner.

11. From the *MainWindow* or *LayoutWindow* use **File → Close Design** to close the current design in the *LayoutWindow*. If the *Close Design* dialog box appears, click on **Discard All** to close the design without saving it.

12. You are done using the GUI. To close the GUI, while keeping the IC Compiler session active, type:

```
stop_gui or gui_stop
```

The *MainWindow* is now closed, but the IC Compiler *shell* is still active in the *UNIX* window.

Task 5. Getting Help with Commands and Variables

1. IC Compiler supports command name, variable name, file name and command option "completion" through the [**Tab**] key. Try the following in the IC Compiler command shell window:

```
h[Tab]e[Tab] –v[Tab] help[Enter]
```

2. To view the *man* page on a command or variable you need to enter the exact command or variable name. Alternatively, you can enter the starting characters of the command/variable and use *command completion* to find the rest. If you are not sure what the exact name is, use `help` for commands, and `printvar` for variables, along with the * wildcard. Here are some examples:

Let's say you are looking for more information about a certain optimization command. You don't remember the exact command name, but you know it contains the string "syn" (for "synthesis"). To list all commands that contain this string enter:

```
help *syn*
```

From the displayed list of commands, you pick out the one you are interested, namely, `psynopt`.

3. To list the available options for `psynopt`, use the `verbose` option:

```
help –verbose psynopt
or
help –v psynopt
```

4. To get a full help *manual page* – a detailed description of the command and all of its options, type:

```
man psynopt
or
man psyno[Tab]
```

5. Now let's say you need help on a specific variable, but again, you don't remember its exact name, but it contains "library". To list all variables containing this string, enter:

```
printvar *library*
```

From the list you identify the variable of interest, namely `target_library`.

Notice that the `printvar` command also lists the current value of each variable.

6. To get a full help manual page of the variable, type:

```
man target_library
or
man target_l[Tab]
```

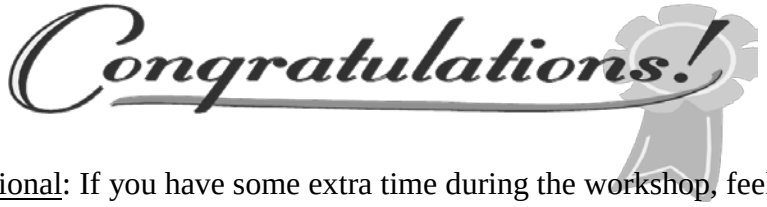
7. Lastly, you can also get additional help with an error or warning message, using the unique message code, for example:

```
man PSYN-025
```

8. Quit the IC Compiler shell:

```
exit    or    quit
```

You have completed the IC Compiler GUI lab 0A.



Lab 0B is optional: If you have some extra time during the workshop, feel free to return and try out the additional GUI features shown in Lab 0B.

Answers / Solutions

Question 1. What command was executed to open the *placed* cell? (scroll up until you find it)

```
open_mw_cel placed
```

Question 2. What is the difference between the “magnifier” button with “2” in it and the button with a “+” in it?

“+” allows you to select a window to zoom into using the mouse. The “2” button magnifies 2x around the center of the current display.

Question 3. What is the difference between the **Vis.** (visibility) and **Sel.** (selection) columns in the above panel?

Visibility is used to turn the display of objects on/off. Selection is used to control which objects are selectable when clicking on them. Invisible objects cannot be selected.

Question 4. On what layer name and number are the red horizontal power straps?

The layer name is METAL3, corresponding to layer number 22. Power nets are defined during design planning.

Question 5. On what layer name and number are the green vertical power straps?

The layer name is METAL4, corresponding to layer number 26.

0B

OPTIONAL: More IC Compiler™ GUI

Learning Objectives

This lab has two purposes:

1. Explore additional features of the IC Compiler GUI
2. Perform GUI-based timing analysis.

You will work with a design that has been previously placed by IC Compiler.

After completing this lab, you should be able to:

- Use the pan/zoom history features
- Configure the windows
- Create and manipulate selection lists
- Highlight layout objects
- Invoke the query tool
- Analyze timing paths
- Cross-probe between the layout and schematic

Lab Duration:

Instructions

Task 1. Window Configuration

1. Change your current directory to *lab0_gui* and invoke the IC Compiler GUI:

```
UNIX$ cd lab0_gui
UNIX$ icc_shell -gui
```

2. Load the ***placed*** cell from the ***risc_chip.mw*** design library:

File → Open Design … or  .

There are multiple windows within IC Compiler, for example *MainWindow* and *LayoutWindow* (there are more). These top-level windows can have multiple "views" or sub-windows in them - by default, the *LayoutWindow* contains an "Overview" (miniature cell view) window and a "View Settings" (layer visibility) panel. You can configure these sub-windows and views any way you like. The sub-windows can be undocked and can "float" anywhere on your desktop, or can be docked in another location of the window (top, bottom, left or right). We'll show you how to do this next.

3. Undock the *View Settings* panel by right clicking over the top edge of the panel (over the two horizontal lines), and selecting "**Float**". The view is now stand-alone (floating or undocked) in its own window. Left-click at the top of the floating panel and drag to move it wherever you like.

4. Dock the panel again by right clicking its top edge and selecting **Dock → Left**.

5. Move the position of the *Context* window down, below the *View Settings* panel, by left-clicking its top banner to "grab and drag" the window down. Release the mouse button when the window is below the *View Settings* panel. You can also use this method to undock and re-dock a window.

6. Close both the *Overview* and the *View Settings* windows by clicking their "Hide" icon  in the top right corner or by right-clicking the top edge and selecting "**Hide**". The windows are hidden. This gives you maximum viewing area for your layout.

7. Re-open the *Overview* and *View Settings* windows as follows: Right-click anywhere in the tool bar at the top of the *LayoutWindow* (where the pull-down menu selections are listed). A menu appears listing the available tool-bar views, followed by the four window views. Select "**Overview**". The *Overview* window appears on the left. Open the *View Settings* panel by either pressing the [F8] hot key or repeating the above step and selecting **View Settings**.

8. From the tool bar select the "**Window**" pull-down menu. Near the bottom is a list of all open windows. Selecting one brings that window to the foreground.

   You can do the same thing with [**Ctrl** `] - Control plus "back-tick" (usually below the "tilde" ~).

9. You can have multiple "layout views" displayed in one *LayoutWindow*. Open a new layout view by selecting **View → New Layout View**. The new view is displayed, along with two "tabs" at the bottom, labeled *Layout.1* and *Layout.2*. You can display the different views by selecting the tab, or by setting up "cascaded" or "tiled" views (**Window → Tile Views** or **Cascade Views**).

10. Maximize one of the two layout views. Optionally, close the other.

    Note: Your last window configuration is automatically saved by IC Compiler and will be restored during your next session. The window configuration is saved when exiting the tool in `~/.config/Synopsys/icc_shell.conf`.

Task 2. Pan and Zoom History

The *LayoutWindow* maintains a history of pan/zoom views. A view can also be saved and later recalled. This can be useful while debugging or analyzing results of large designs.

1. If the "history buttons" shown here do not already appear in the *LayoutWindow* tool bar, right-click in the tool bar and select "**Zoom and Pan History**". Three additional "history" buttons are added to the tool bar.

2. If you do not like the default location of the tool-bar buttons, you can change it. Tool-bar button "groups" can be moved to the left or right by left-clicking over the double-vertical bars and dragging and dropping in the desired area.

3. Zoom in on an area of interest [**Z**], then **[Esc]**.

4. Click on the right-most "history" button shown above  or select the menu entry **View → Zoom → Named Zoom and Pan Settings…** .

   In the dialog box enter the name "*myzoom*", click **Add** and then **Close**.

5. Return to a full view, by typing lower-case [**F**] key.

6. To retrieve the saved view, bring up the same dialog box, select "*myzoom*" and click on **Zoom To**. **Close** the dialog box.

7. Pan or zoom to several different areas of interest on the layout and then try the "Go Back" and "Go Forward" buttons  to cycle through the view history.

Task 3. Selection Lists, Highlighting and Querying

1. Selection lists:

Zoom into the blue core area of the layout and select four or five standard cells.

2. Reduce the *brightness* to **50%** to improve the contrast. The "***Brightness***" control is located at the top of the *View Settings* panel.

3. Show the selected objects in a list format: Use the **Select → Selection List** menu entry.

4. The list can be further filtered by using the **Select**/**Deselect** buttons: Using the [**Shift**] or [**Ctrl**] key, select all except the first two objects from the selection list and click the **Deselect** button to remove them from the list. Notice that only those two standard cells remain selected in the *LayoutWindow*.
Keep the selection list open, and the standard cells selected.

5. Highlighting objects:

In the tool bar locate the solid yellow "color" rectangle  . Click on the pull-down menu and select the color *red* , then click on the "pen" button to apply the highlight to the selected cells. If the selection list disappeared bring it back with [**Ctrl L**]. To see the colored highlights you must unselect the cells ([**Ctrl D**] or click an area with no objects). The objects remain highlighted in red.

Select one other cell that is not currently highlighted.

Question 1. Does the selection list contain the highlighted objects?

.....

Click the **Close** button on the *Selection List* window to close it.

Clear the highlighted objects by clicking  or [**Ctrl M**].

6. Querying objects:

By default, when the cursor arrow hovers over an object, the object is lightly highlighted, and a query "summary" window appears, displaying some key attributes of the object.

To obtain a "full query", select a single standard cell, and query it by typing lower-case [**Q**] or by using the menu entry: **Select → Query Selection**. A window opens and lists all the cell's attributes and attribute values. You may need to expand the window to the left to see the values of each attribute.

7. Close the query window by clicking the "Hide" **minus sign** in its upper right corner.

8. If you do not want to change the current set of selected items, or you want to query many objects, there is an alternative method. Turn on the "Query tool" either by clicking on the  button, by pressing [**Ctrl Q**], or by using the menu item **View → Mouse Tools → Query Tool**. The cursor image changes to  .

Now click on any object and you will see information displayed in the "Query Objects" window, without affecting the currently selected objects.

Close the "Query Objects" window. Exit the query tool by pressing [**Esc**].

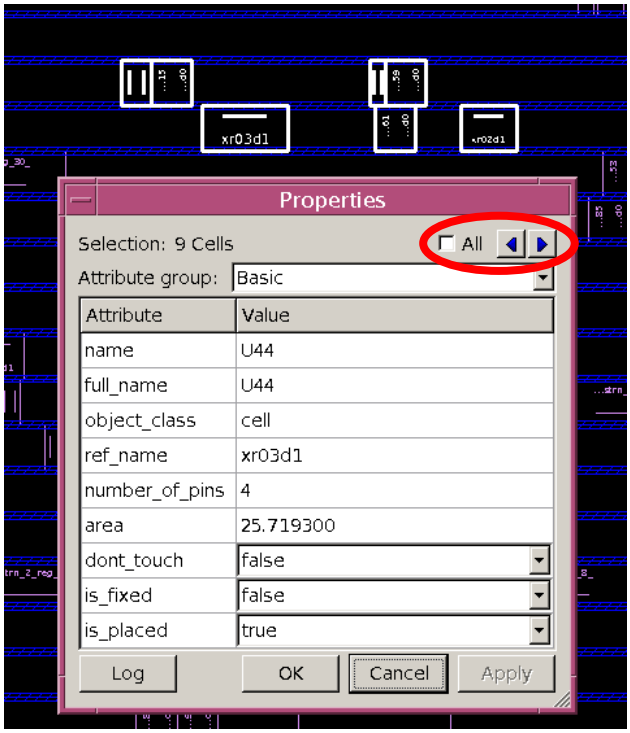
9. Select a few standard cells again. A final way to look for information in a list of selected cells is with the menu entry **Edit → Properties** or [**Ctrl R**]. This will bring up the *Properties* dialog box shown below.

This dialog can also be used to change properties of selected objects.

Note: You can also bring up the *Properties* dialog box by right clicking the core area and selecting **Properties** in the pop-up menu.

10. Click the right and left blue arrow buttons to cycle through the properties of each item in the selection one at a time. Check the "**All**" box to see the properties that are identical for all the selection items.

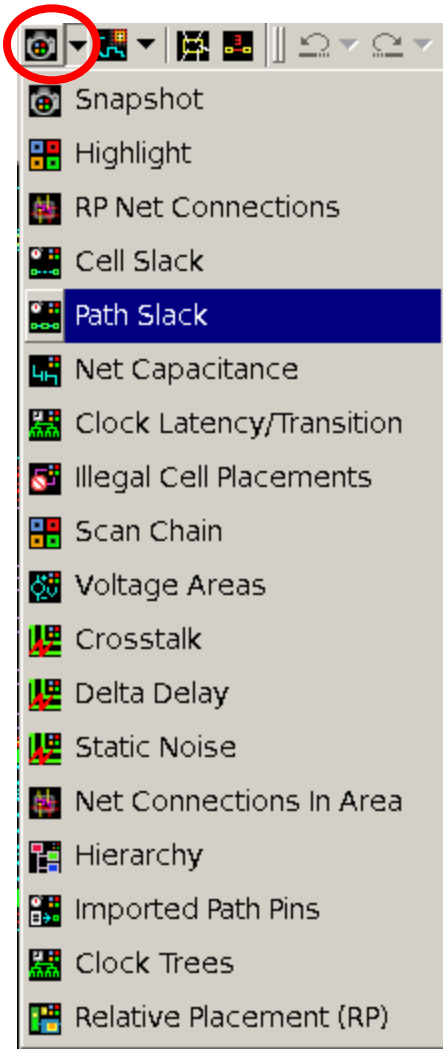
11. **Cancel** the *Properties* dialog box, unselect all objects [**Ctrl D**], and fit the layout view to the window [**F**]. Return the "brightness" to **100%**.



Task 4. Analyze Timing Paths

IC Compiler contains many advanced and easy to use timing analysis and visualization features. We will discuss some useful ones here.

1. Highlighting critical paths:



Select the pull-down menu shown on the right and click on **Path Slack**.

2. In the dialog that appears, click on **Reload**.

3. Click **OK** in the *Warning* box.

4. Wait a little for the *Paths Slack* dialog box to open, then click **OK**.

You should now see colored "fly-lines" in your layout window. The most critical paths (least slack) are highlighted in pink, then red, etc. The most critical path is the one that ends at the IO pad named *RESULT_DATA_iopad_0* in the lower right. **Keep the paths highlighted**.

5. Cross-probing between the layout and schematic:

Select the IO pad mentioned above, ***RESULT_DATA_iopad_0***.

6. Bring the *MainWindow* to the foreground [**Ctrl** `] and maximize it.

7. Select **Schematic → New Path Schematic View → Of Selected Logic**. A schematic window opens, showing just the selected cell – an IO pad "buffer".

8. Double click the top bar of the schematic window to enlarge the view, and fit [**F**] the schematic to the window.

If not already selected (white), select the displayed gate, then right click and select "**Next Fanin/Fanout Level**". Use [**F**] to fit the schematic to the window. You should see one additional level of fanin and fanout to/from the single gate.

Now select the new gate fanning in to our original gate. Right click and select "**Next Fanin/Fanout Level**". You will see an additional level of fanin added. Instead of selecting the gate, you can also just select one of the pins (arrows) and perform the same function. This is useful if you want to expand the path through one particular input pin of a multiple-input gate. You can also double-click the input or output pin to display the next fanin or fanout level. During actual analysis you would continue expanding the path to meet your needs.

9. Unselect all objects in the schematic by clicking in the black background, away from any object, or by pressing [**Ctrl D**]. Notice that there are no selected (white) objects in the layout view.

10. In the schematic, select the original right-most buffer ***RESULT_DATA_iopad_0***. Notice in the layout view that the same IO pad cell is selected (highlighted in white).

11. In the schematic add the fanin nets, pins and gates to your selection using either [**Ctrl**], or by dragging a left-mouse rectangle around them. Notice that these objects are also highlighted in the layout.

12. In order to see this entire critical path, select just the original gate in the schematic, right click and select **Add Logic → Paths…** In the dialog that appears, simply select **OK**. This will display the worst (critical) path to the selected logic gate.

13. Generating a schematic for the entire design:

In the *MainWindow* press [**Ctrl D**] to clear all selections, then select the menu **Schematic → New Design Schematic View**.
A new schematic window opens containing the top-level schematic of the current design, *placed*.

14. The schematic contains a single hierarchical instance *I_RISC_CORE,* plus a handful of logic gates.

Click on the rectangular *I_RISC_CORE* bounding box to select it (located in the left part of the schematic). Check that the "*Cell*" name that appears in the lower right corner confirms that *I_RISC_CORE* is selected. Double click inside the *I_RISC_CORE* boundary to traverse down the hierarchy and view its schematic.

15. Use the *Up Arrow*  and return to the top schematic level.

16. Using the *TimingWindow* for timing analysis:

Let's examine the overall timing quality of the design by displaying a list of path slacks for analysis.

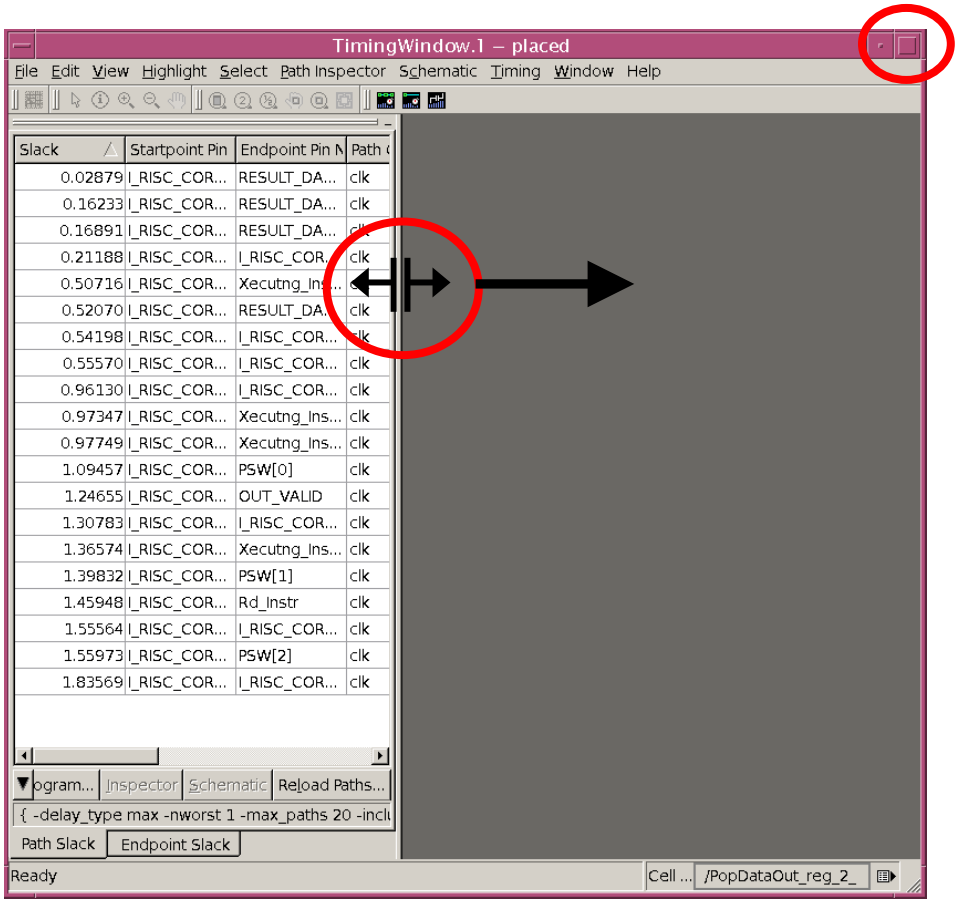
In the *MainWindow* select **Window → New Timing Analysis Window…** .

Click "**OK**" in the *Select Paths* dialog box.

A new window, labeled *TimingWindow* opens, in addition to the two windows already open.

17. Maximize the *TimingWindow* by double-clicking the title bar, or by selecting the “square” button in the upper right corner.

Expand the *slack list* window to the right by placing your cursor arrow over the right border of the window. The cursor changes shape (show below). Click the left mouse button and drag the boundary to the right.

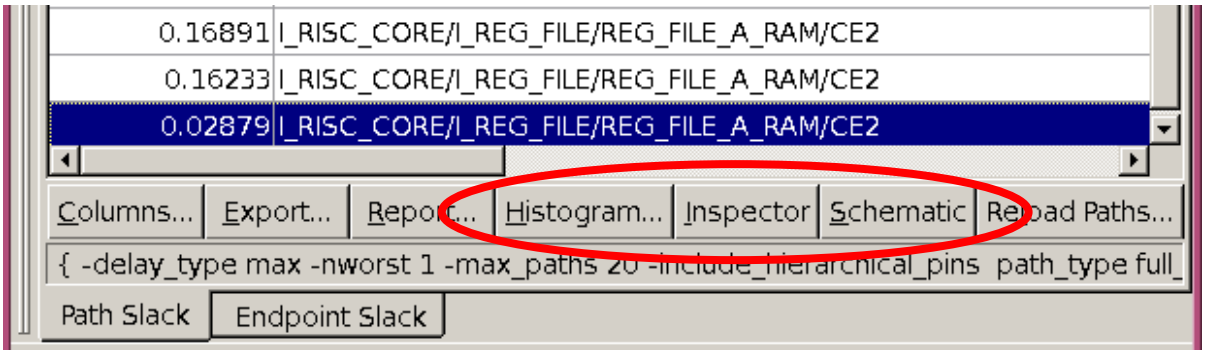


You can left click and drag the gray column header boundaries to resize the column widths if the field values are not fully displayed. You can re-arrange the order of the columns by left-clicking a column header and dragging it to the desired location. Click inside a gray column header and a *sort arrow* appears allowing you to sort the column’s data.

18. Select the path with the smallest *Slack* in the list by left-clicking it. The line should highlight to a blue color.

Examine the *LayoutWindow* and *MainWindow* and observe that the path is highlighted in white.

19. There are controls for generating histograms, schematics and timing profilers at the bottom of the *TimingWindow*:

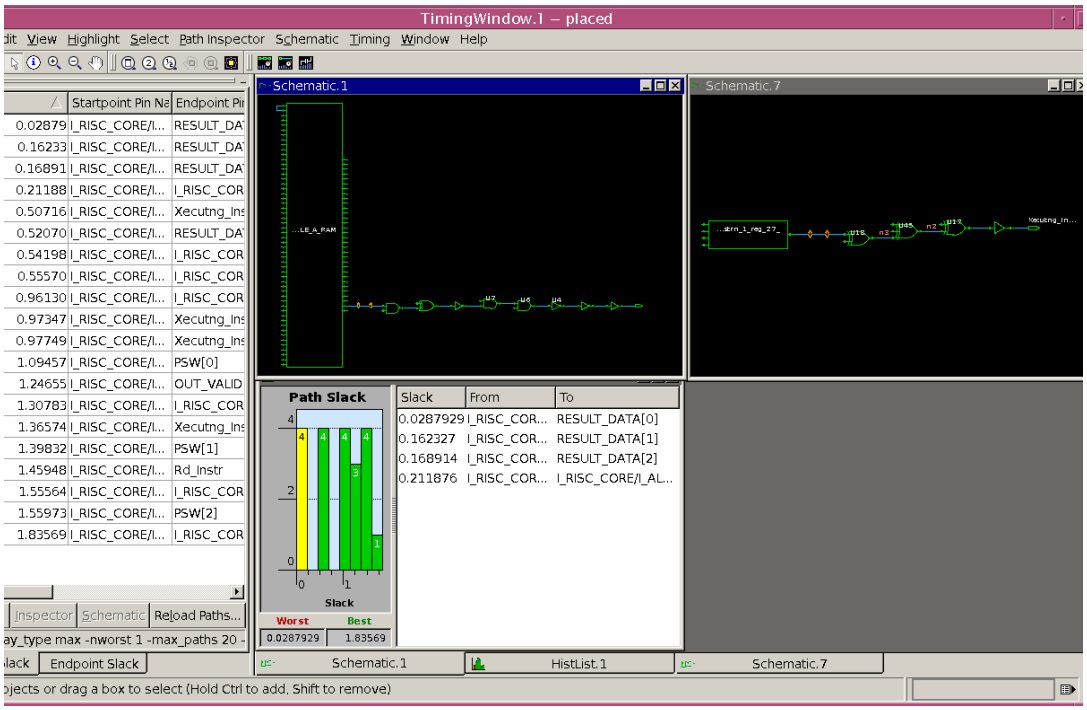


Click the ***Schematic*** button and a schematic of the highlighted path appears in a new window in the gray area of the *TimingWindow*, or in a new tab. You can select and open multiple schematic windows (two are open in the example below and have been re-arranged to fit next to each other).

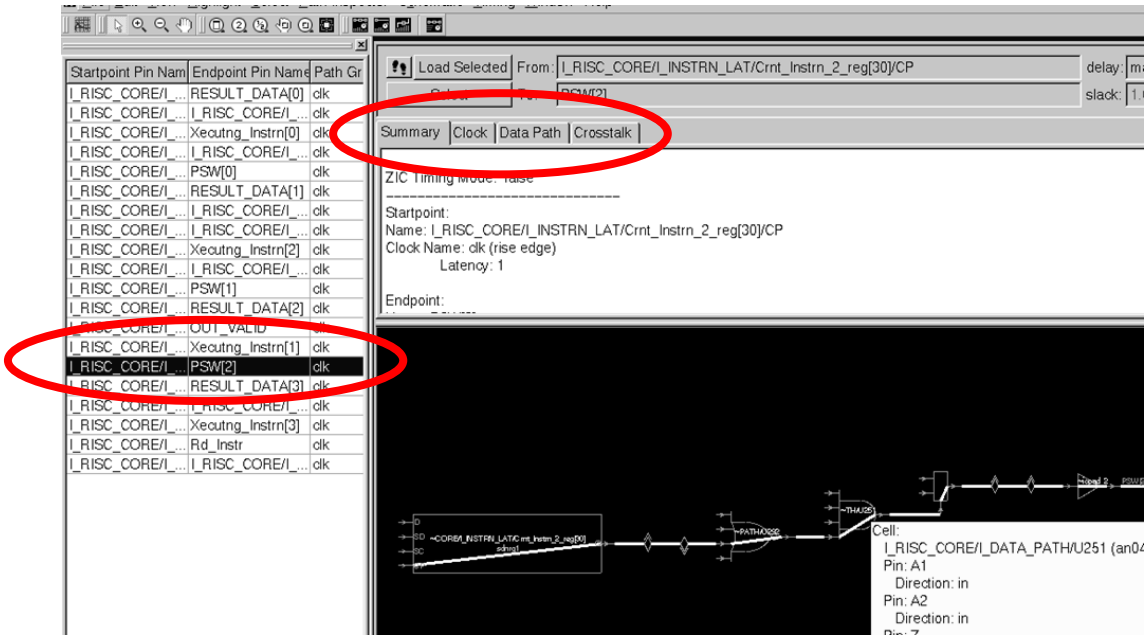
Click the ***Histogram*** button and "**OK**" the dialog box. A set of timing slack histogram bars appears. Select one of the histogram bars and the list window populates with all the paths in that bar. Selecting one of these paths updates the selection in the master path slack list and the path display in the *LayoutWindow*.

Note: The histogram tool is similar to that available from the Design Compiler and PrimeTime GUIs.

Note: To clear a highlighted path, select [**Ctrl D**] as usual.



The *path inspector* can be started by pressing the "**Inspector**" button at the bottom of the *TimingWindow*.
Note: You must first select a path that you wish to "inspect" from the list on the left.



This also opens a schematic window, along with another window that can display details about the path (by selecting the appropriate tab): The *Clock* tab lists details about the launching and capturing clock of the path; *Data Path* lists timing and additional details along the entire path; Crosstalk lists related delay and parasitic information.

Task 5. Window Management

1. A final note on window management: It is possible to have multiple *MainWindows*, *LayoutWindows* and *TimingWindows* open at the same time. You can also open multiple designs at the same time, and switch between them.

If you have not already done so, try the [**Ctrl `**] (control back-tic) sequence to cycle through these windows, similar to an ALT-TAB in an *MS Windows* interface.

If you have multiple tabs (or views) within a window, you can cycle through them using [**Ctrl Tab**].

If you have problems with windows disappearing, e.g. a *Properties* window that vanishes under a *LayoutWindow*, check the controls in your Linux/Unix window manager. Find a control to keep "Secondary windows" on top.

2. From the *MainWindow* or *LayoutWindow* use **File → Exit → Discard All** to exit IC Compiler.

You have completed the optional GUI lab.

Answers / Solutions

Question 1. Does the selection list contain the highlighted objects?

The list tracks what is selected at any time, not the highlighted objects.

1

IC Compiler™ Data Setup & Basic Flow

Learning Objectives

This lab has two purposes:

1. Walk you through the "data setup" process of creating and maintaining a *Milkyway* database to hold your design data.
2. Run a complete basic flow, from loading a floorplan through routing.

After completing this lab, you should be able to:

- Create a *Milkyway* database for your design
- Attach *reference libraries* to your design library
- Load *TLU+ models* for accurate parasitics modeling
- Read in a netlist
- Apply *sdc* constraints
- Apply timing and optimization controls
- Load a *DEF* format floorplan
- Place and optimize the design using `place_opt`
- Build and optimize a clock tree for the design using `clock_opt`
- Route and optimize the design using `route_opt`
- Generate and interpret a timing report
- Load a previously saved design in a new session

Lab Duration:

Introduction

In this lab you are provided with netlist, timing constraints and floorplan data for a design called *RISC_CHIP*. You will create a *Milkyway* design library from the provided design data in the first part of the lab. In the second part of the lab, you will place the standard cells, create the clock tree and route the *RISC_CHIP* design using the basic flow.

This design is very simplistic and is only meant as a vehicle for observing the basic flow.

Answers / Solutions

There is an *ANSWERS / SOLUTIONS* section at the back of each lab. You are **encouraged** to refer to this section to verify your answers.

Relevant Files and Directories

All files for this lab are located in the ***lab1_data_setup*** directory under your home directory.

`lab1_data_setup/`

`.synopsys_dc.setup`	Read by IC Compiler upon startup
`design_data/`	
`RISC_CHIP.v`	RISC_CHIP verilog gate level netlist.
`RISC_CHIP.def`	RISC_CHIP floorplan in DEF.
`RISC_CHIP.sdc`	RISC_CHIP timing constraints.
`scripts/`	
`opt_ctrl.tcl`	Timing and optimization controls.
`zic_timing.tcl`	A script used to check zero-interconnect timing constraints.
`derive_pg.tcl`	Create logical P/G connections.
`.solutions/`	
`run.tcl`	A run script with all the commands executed in this lab.

Instructions

Task 1. Create a *Milkyway* library

1. Change your current directory to *lab1_data_setup* and look at the contents of the directory.

```
UNIX% cd ../lab1_data_setup
UNIX% ls –a
```

You should see the directories listed on the previous page, and the `.synopsys_dc.setup` file.

2. Use a UNIX text editor or viewer to look at the contents of the `.synopsys_dc.setup` file.

3. At the bottom of the file, we have created the following user-defined variables to help document and simplify the data setup process:

```
#--------------------------------------------------
# RISC_CHIP setup variables
#--------------------------------------------------
set my_mw_lib risc_chip.mw
set mw_path “../ref/mw_lib”
set tech_file “../ref/tech/cb13_6m.tf”
set tlup_map “../ref/tlup/cb13_6m.map”
set tlup_max “../ref/tlup/cb13_6m_max.tluplus”
set tlup_min “../ref/tlup/cb13_6m_min.tluplus”
set top_design   “RISC_CHIP”
set verilog_file “./design_data/RISC_CHIP.v”
set sdc_file     “./design_data/RISC_CHIP.sdc”
set def_file     “./design_data/RISC_CHIP.def”
set ctrl_file    “./scripts/opt_ctrl.tcl”
set derive_pg_file “./scripts/derive_pg.tcl”
```

If you lose track of what these variables contain as you proceed, you can easily query them within the *icc_shell* using the `printvar` command.

4. The section above the user-defined variables contains the *logic library* settings which were discussed in the lecture:

```
lappend search_path ../ref/db ../ref/tlup
set_app_var target_library “sc_max.db”
set_app_var link_library “*sc_max.db io_max.db \
                      ram16x128_max.db”
set_min_library sc_max.db -min_version sc_min.db
set_min_library io_max.db -min_version io_min.db
set_min_library ram16x128_max.db -min_version \
                             ram16x128_min.db
```

Above that we have defined some aliases which will be used later in this lab. There are also settings that control the creation of log files.

Note: These tool variables can be applied in any order, not necessarily the order shown here.

5. Exit the text editor or viewer.

6. Start IC Compiler from the UNIX prompt:

```
UNIX% icc_shell
```

IC Compiler starts in the *xterm* window. All output is also logged to the file `icc_shell.log.*` . This logging was configured in the `.synopsys_dc.setup` file.

7. Verify that the `.synopsys_dc.setup` file was indeed read in, by querying one of the user-defined variables:

```
printvar sdc_file
```

8. Start the GUI. The *MainWindow* will appear after a short while:

```
start_gui
```

Note: Or just type “**gui**”, a workshop-provided Tcl-procedure !

9. Create the design library:

a. Use the *MainWindow* menu **File → Create Library …** to bring up the *Create Library* dialog box.

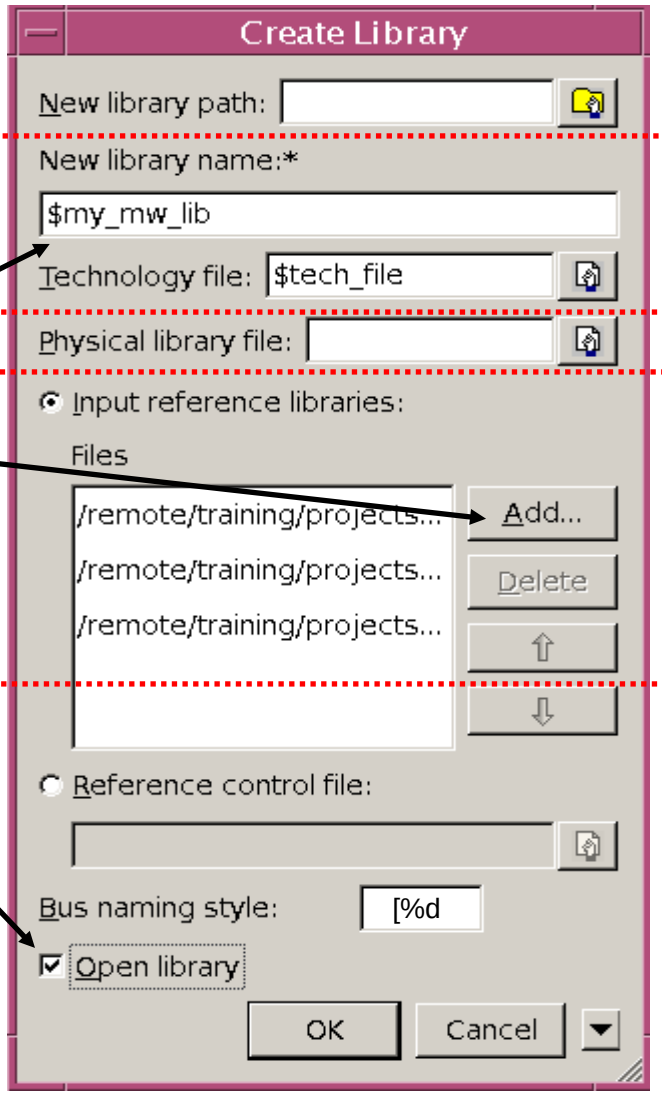
b. Use the *variables* already defined for the library and tech file names.

c. Attach *reference libraries* to your design library:
Click the **Add…** button.
Double-click .. to move up one level, then double-click ***ref*** followed by ***mw_lib***.
Select the standard cell library "***sc***" and click **OK** to add it to the list.

d. **Add** the "***io***" and "***ram16x128***" libraries as well.

e. Select the "**Open library**" check box to open the design library after it is created.

f. Click **OK**.



Note: The ***Warning*** about missing "CapModel sections" is expected. We will load TLU+ files later.

The following command is the equivalent of the GUI operation above:

```
create_mw_lib -technology $tech_file -mw_reference_library \
              "$mw_path/sc $mw_path/io $mw_path/ram16x128" \
             -bus_naming_style {[%d]} -open $my_mw_lib
```

10. Type the following in another *xterm* window, or in the *IC Compiler shell*, and note the contents of the newly created *UNIX* directory `risc_chip.mw` (the design library).

```
UNIX% ls -a risc_chip.mw     OR
icc_shell> ls risc_chip.mw
```

Note: You should see three *lib** files and a *.lock* file.

Task 2. Load the Netlist, *TLU+*, Constraints and Controls

1. Before reading in the Verilog netlist make sure the design library is open: An easy way to do so is by checking if the **File → Open Library …** entry is grayed out. If it is, this confirms that a design library is currently open.

2. Select **File → Import Designs …** to bring up the *Import Design* dialog box.

3. Under *Input format* select ***verilog***.

4. Click **Add …** then browse to select the file **`design_data/RISC_CHIP.v`** and **Open**.
Under *Top design name* enter **`$top_design` (or `RISC_CHIP`)**.
Click **OK**.

The following command is the equivalent of the GUI operation above:

```
import_designs $verilog_file -format verilog \
                             -top $top_design
```

The *Verilog* netlist is read in and a *LayoutWindow* opens, displaying all the netlist cells stacked on top of each other at the origin. The larger IO pad and macro cells are shown in light blue. The much smaller standard cells are in light purple (you may need to zoom in to the lower left corner to see them).

Question 1. How has the UNIX content of the design library changed?

.....

.....

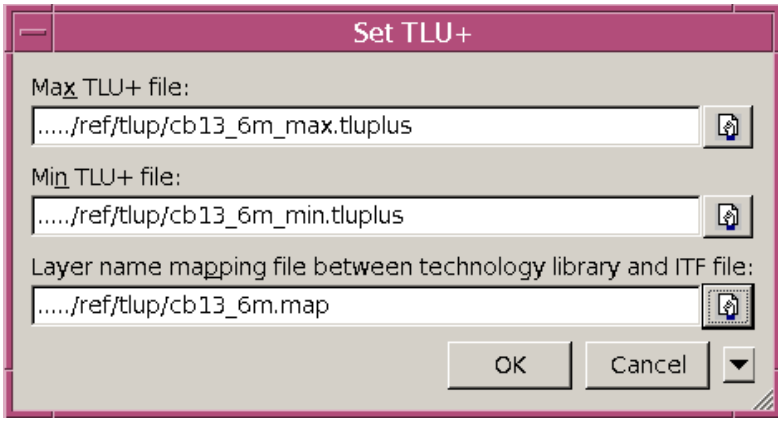
.....

Check your answer against the answer at the end of this lab.

5. From the *MainWindow* use the menu **File → Set TLU+ …** to bring up the *Set TLU+* dialog box.

Click the  *browse* button, then double-click  as needed to locate the ***ref*** directory. Enter paths to the files shown below.

Click **OK** to load the *TLU Plus* parasitic files.



Note: The following command is the equivalent of the GUI operation above:

```
set_tlu_plus_files -max_tluplus $tlup_max \
                   -min_tluplus $tlup_min \
                   -tech2itf_map $tlup_map
```

6. Check the physical and logical libraries for consistency:

Note: We will do a default check instead of the recommended complete check (`set_check_library_options –all`), which would include checks related to *UPF power constraints*, *multi-corner multi-mode*, and *CCS current modeling*. These checks would generate many warnings and errors that do not apply to our libraries.

```
check_library
```

There are two messages that are of interest:

“Number of cells missing in logic library : 19”

This message lists *feed-through*, *power pad* and *substrate tap* cells. These cells are only used in the physical layout and are therefore not needed in a “logic” library. This message can be safely ignored.

“Number of cells with missing or mismatched pins in libraries: 12”

This message points out that the listed cells have mismatched “pin types”. The *logic* library lists them as type “`signal`”, while the *physical* library lists

them as "ground" or "power". These cells are also physical-only cells, and since the physical library types are correct, we can safely ignore this message as well.

Note: If the above check were to list any missing or mismatched standard cells, macro or IP cells, or IO pad cells, then the libraries would potentially need to be modified to correct the situation.

7. Check that *TLU+* files are attached and that they pass three sanity checks:

```
check_tlu_plus_files
```

You should see a listing of the files for `max_tlu+`, `min_tlu+` and `mapping_file` and all *sanity checks* should say **`[Passed!]`**.

8. Verify that the specified link libraries have been loaded:

```
list_libs
```

Note: You should see the six logic (*db*) libraries (max and min) that were specified by `set_app_var link_library` and `set_min_library`, as well as two "generic" logic libraries, which are always loaded by default: *gtech.db* and *standard.sldb*.

9. Define the "logical" connections between power/ground pins and nets.

```
source $derive_pg_file
check_mv_design -power_nets
```

There should be no "unconnected" power or ground pins.

Note: The contents of the above file is shown below. Besides VDD and VSS, this design has two additional P/G signals that are used in the periphery area: VDDO/VSSO, and VDDQ/VSSQ:

```
derive_pg_connection -power_net VDD -power_pin VDD \
                     -ground_net VSS -ground_pin VSS
derive_pg_connection -power_net VDDO -power_pin VDDO \
                     -ground_net VSSO -ground_pin VSSO
derive_pg_connection -power_net VDDQ -power_pin VDDQ \
                     -ground_net VSSQ -ground_pin VSSQ
derive_pg_connection -power_net VDD -ground_net VSS -tie
```

10. Apply the top level design constraints:

```
read_sdc $sdc_file
```

The next several commands are recommended to verify key constraints, or to get specific information about key constraints.

11. Check if any key timing constraints (for example clocks, input/output constraints) are missing:

```
check_timing
```

Note: The check should not give any “Warning” or “Error” messages. “Information” messages followed by a “1” means that there are no missing or inconsistent constraints.

12. Check to see what “timing exception” constraints are applied to your design. These include *false* and *multicycle* paths, as well as asynchronous *min*- and *max-delay* constraints. These constraints are an “exception” to the default “single-cycle” timing behavior – it is useful to know if your design contains any of these timing exceptions, and where they are being applied:

```
report_timing_requirements
```

Note: There should be no timing exception constraints reported.

13. Check to see if timing analysis was disabled along any paths. If disabled timing arcs exist, you would probably want to check with the synthesis group if they are still required during the physical design phase:

```
report_disable_timing
```

Note: Since no paths are listed there are no disabled timing paths.

14. Check to see if the design has been configured for a specific “mode” or “case”, for example “functional” versus “test” mode. This is done by constraining a control pin or port to a constant *logic 0* or *1* during timing analysis and optimization only, not “hard-wired”. This is helpful to confirm if your design is in the correct “mode” for physical design optimizations:

```
report_case_analysis
```

Note: Since no pins are listed there are no constants applied.

15. Verify that the clocks are appropriately modeled:

```
report_clock
report_clock –skew
```

Note: The `report_clock` output confirms that the clock is not "propagated" (otherwise there would be a "p" in the Attributes column). This is what you want prior to clock tree synthesis. The *clock_skew* report confirms that clock tree effects (*insertion delay*, *skew*, *transition time*) are being modeled.

Question 2. What is the combined modeled effect of skew, jitter and margin for setup time?
(Hint: "Minus Uncertainty" relates to *setup* time)

.....

16. Apply some timing and optimization controls which are specified in *./scripts/opt_ctrl.tcl*:

Note: Most of these settings are discussed in the *Appendix* of Unit 1. Some are discussed in later Units. **Do not spend time here trying to understand them:**

```
source $ctrl_file
```

17. Run a "zero-interconnect" (*zic*) timing report. Recall from the lecture that the *ZIC* mode sets the capacitive load of wires to zero. Try using "file name completion" inside the `icc_shell`, using the [**Tab**] key.

```
source sc[TAB]z[TAB]
# The above file scripts/zic_timing.tcl contains:
#     set_zero_interconnect_delay_mode true
#     redirect –tee zic.timing { report_timing }
#     set_zero_interconnect_delay_mode false
```

18. The above `redirect -tee` command displays the timing report on the screen *and* saves it to a file. You can look at the contents of that file by executing a UNIX "`cat`"at the `icc_shell` prompt:

```
exec cat zic.timing
```

Scroll up and look at the entire timing report. There are three paths listed – one for each applicable "Path Group" called *INPUTS*, *OUTPUTS* and *clk*.

These *path groups* were defined in the timing/optimization control file that was applied a couple of steps above, and will be explained in a later unit.

Question 3. Does the constrained design pass the "ZIC" sanity check?

.....

19. The "scan enable" signal (`scan_en`) was defined as an *ideal network* (see `$sdc_file`) to prevent synthesis from buffering this signal. Remove the ideal network definition so that it will be buffered during physcial design:

```
remove_ideal_network [get_ports scan_en]
```

20. Save the cell and notice the new binary files under *risc_chip.mw/CEL*:

```
save_mw_cel –as RISC_CHIP_data_setup
```

Congratulations! You have completed "data setup". In the next few tasks you will take your design through the very basic steps of *design planning*, *placement*, *clock tree synthesis* and *routing*. You will explore each of these design phases and associated commands in much greater detail in the upcoming workshop Units.

Task 3. Basic Flow: Design Planning

For this lab we have provided you with a predefined floorplan file, in standard *DEF* format. This file can be generated by IC Compiler, after the design planning phase, or by a third party design planning tool.

1. Read in the provided *DEF* file:

```
read_def $def_file
```

Note: You may also use the GUI: **File → Import → Read DEF…** to read **`./design_data/RISC_CHIP.def`**.

2. Press [**F**] in the *LayoutWindow* to refresh the view. You should now see the floorplanned design.

3. Ensure that standard cells will not be placed under the power and ground metal routes (this constraint is not part of *DEF*):

```
set_pnet_options -complete {METAL3 METAL4}
```

4. Save the design cell and notice the new binary files under *risc_chip.mw/CEL*:

```
save_mw_cel –as RISC_CHIP_floorplanned
```

Task 4. Basic Flow: Placement

1. Place and optimize the design for timing, and generate a timing report:

```
place_opt
redirect -tee place_opt.timing {report_timing}
```

Question 4. Does the placed design meet timing?

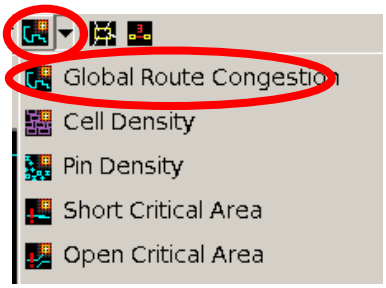
.....

2. In the *LayoutWindow*, zoom in and take a look at the standard cell placement.

Question 5. Would you call this a "core limited" design?

.....

Analyze congestion :
Select the dropdown menu to the right of the  icon. If you don't see the icon, click on the **>>** to extend the tool bar.



Select "**Global Route Congestion**" from the dropdown, then select "**Reload**".

A dialog box appears, which contains the command to be executed for congestion analysis. In the "*(Re)Calculate Global Route Congestion Map Data*" field, make sure that the command below is listed:

```
report_congestion -grc_based -by_layer \
    -routing_stage global
```

Click "**OK**".

The congestion "heat map" is shown in the layout. There are 37 edges of 0 "overflow", which means that there are just enough routing tracks for the required metal traces.

3. Close the congestion map by clicking on the small "**x**" in the upper right corner of the congestion dialog box.

4. Save the design cell:

```
save_mw_cel –as RISC_CHIP_placed
```

Task 5. Basic Flow: CTS

1. You will be using default settings to generate the clock tree. However, in order to allow IC Compiler to calculate the actual clock skews during clock tree synthesis, instead of incorporating the estimated skew from the constraints, remove the "*clock uncertainty*" first. Also, enable hold-time fixing. We will discuss this in greater detail in the CTS unit.

```
remove_clock_uncertainty [all_clocks]
set_fix_hold [all_clocks]
clock_opt
redirect -tee clock_opt.timing {report_timing}
```

Question 6. Is the timing still OK?

.....

2. Display the clock tree: Use the *LayoutWindow* menu **Clock → Color By Clock Trees** to bring up the "visual mode" dialog box.

Click "**Reload**".

In the dialog box that appears, make sure the *Source Pin Name* "**clk**" is selected (highlighted in blue).

At the bottom of the dialog box select the box "**All Levels, Types**".

Click **OK**.

The clock tree metal interconnects (or routes), as well as the standard cells, IO pad and macro cells connected to the clock tree, are highlighted. Notice how the clock tree starts at the IO pad cell "**clk_iopad**" (top edge of the periphery, on the right), then connects to all the registers ("**sdnrq#**" or "**sdcrq#**") and macro cells (zoom in, or hover your cursor over a cell to see its name).

3. Remove the clock tree highlight by closing ("**x**") the visual mode window.

4. Save the design cell:

```
save_mw_cel –as RISC_CHIP_cts
```

5. We still need to route the design, but first:
Exit IC Compiler by clicking **File → Exit → Discard All**, or typing **`exit`** or **`quit`** and **`OK`** at the ic_shell prompt. We'll explain why we did this next…

Task 6. Basic Flow: Routing

The reason for exiting IC Compiler in the previous step is to have you go through the steps of starting a new session, re-applying specific settings, and continuing from where you left off previously.

1. Invoke IC Compiler's GUI:

```
UNIX% icc_shell -gui
```

2. Since the design library has already been created, and you saved the layout cell after CTS, all you have to do is load the *RISC_CHIP_cts* cell from the *risc_chip.mw* design library, as follows:

a. In the *MainWindow* click on the little yellow "open design" icon  on the top left, or use the menu command: **File → Open Design …**

b. In the *Open Design* dialog panel, click the yellow folder icon  . The *Select Library* dialog box opens. Select the  library folder ***risc_chip.mw*** and click **Choose**.

c. Select ***RISC_CHIP_cts*** and click **OK** to open it.

3. Re-apply the timing and optimization controls, which were applied during data setup. This is required because some of the settings are applied using variables. In general, variable settings are not saved with the design cell – they remain set during the current IC Compiler session. After exiting and re-invoking IC Compiler, the variables are reset to their original default values:

```
source $ctrl_file
```

4. Now we are ready to continue to route the design. This will take care of all the signal nets (the clock nets were already detail-routed by `clock_opt`):

```
route_opt
```

Zoom in and take a look at some of the detailed routing.

5. Generate a timing report. You should see positive slacks:

Note: Since the output of this report can be long we will use a helpful user-created command "view". It opens a separate window in which the output can be conveniently scrolled and searched. The *view* command is a TCL "procedure" that was defined for this workshop. It is not a standard command available in IC Compiler. This procedure, along with several others, is defined at `../ref/tools/procs.tcl`, and is "sourced" in the `.synopsys_dc.setup`.

```
view report_timing -nosplit; # OR use aliases:
v rt
```

Note: Aliases are defined in the `.synopsys_dc.setup` file.

6. By default timing reports show *maximum delay* or *setup* timing. Generate a *min-delay* or *hold* timing report. You should also see that there are no hold violations:

```
v rt –delay min
```

7. Generate physical design statistics:

```
report_design –physical
```

Note: This report includes information about the design's utilization percenatge, congestion (overflow), etc.

8. Save the design.

```
save_mw_cel -as RISC_CHIP_routed
```

9. Quit the IC Compiler shell

```
exit    or    quit
```

You performed all the required data setup steps, and have run the RISC_CHIP design through the basic flow of IC Compiler!

Answers / Solutions

Question 1. How has the UNIX content of the design library changed?

It now contains a *CEL* sub-directory that holds the binary files for *RISC_CHIP* – the top design name that was saved with `import_designs`.

Question 2. What is the modeled combined effect of skew, jitter and margin for setup time?
(Hint: "Minus Uncertainty" relates to *setup* time)

The total effects of (skew + jitter + margin) for setup timing is modeled by a "Minus Uncertainty" of 0.5 ns. During timing analysis (prior to clock tree synthesis), IC Compiler will move all capturing clock edges 0.5ns earlier, thereby reducing the effective clock period by that amount for setup timing.

Question 3. Does the constrained design pass the "ZIC" sanity check?

You will see timing for three different paths reported. The **INPUTS** *path group* contains all the paths between primary input ports and registers; The **OUTPUTS** *path group* contains paths from registers to primary output ports; The **clk** group contains register-to-register paths.

With ZIC timing you expect to see positive slack, or at worst, a small violation. The **OUTPUTS** group has a small negative slack of around 1% of the required path delay. Placement, CTS and/or routing optimizations should be able to fix this violation. Therefore, the netlist ***passes*** the sanity check.

Question 4. Does the placed design meet timing?

Yes. Notice that the small "zero-interconnect" timing violation is gone. All paths have a positive slack now.

Question 5. Would you call this a "core limited" design?

No, it is pad limited – the size of the chip is determined by the large number of pads, not the number of standard cells and macros.

Question 6. Is the timing still OK?

Yes, all paths have positive slack.

2

Design Planning

Learning Objectives

After completing this lab, you should be able to:

- Define the core and placement row structure
- Define locations for signal, P/G and corner pads
- Insert filler pad cells
- Manually place a few macros
- Apply macro placement constraints
- Place macros and standard cells using “virtual flat placement”
- Analyze congestion
- Create power/ground rings around groups of macros
- Complete the power/ground rings and straps using Power Network Synthesis (PNS)
- Analyze IR drop using Power Network Analysis (PNA)
- Analyze timing

Lab Duration: 100 minutes

Introduction

The purpose of this lab is for you to become familiar with the design planning capabilities in IC Compiler. For this lab you will use a slightly larger version of *ORCA*, which is at the chip-level and includes IO pad cells as well as many more macros. This makes the design planning steps more interesting.

Answers / Solutions

There is an *ANSWERS / SOLUTIONS* section at the back of this lab. You are **encouraged** to refer often to this section to verify your answers, or to obtain help with the execution of some steps.

Relevant Files and Directories

All files for this lab are located in the ***lab2_dp*** directory under your home directory.

lab2_dp/	
orca_lib.mw/CEL	
orca_setup	The ORCA design after "data setup", saved in *Milkyway* format.
design_data/	**Contains the ORCA design input data**
ORCA_2.v	Second-pass Verilog netlist
ORCA_2.sdc	Second-pass SDC timing constraints
scripts/	**Contains various floorplanning scripts**
2ns_pass_setup.tcl	Script to perform 2nd pass data setup
connect_pg.tcl	Logically connect all P/G pins to nets
insert_pad_filler.tcl	Insert pad fillers
keepout.tcl	Placement keepout for all macros
macro_place_cons.tcl	Macro placement constraints
macro_pg_rings.tcl	Create P/G rings around macro groups
opt_ctrl.tcl	Timing and optimization controls
pad_cell_cons.tcl	Define pad cell locations
pns.tcl	Power network constraints/synthesis
preplace_macros.tcl	Place three macros connected to IO pads

If you encounter problems or get stuck, a complete command script is available to help you recover: *.solutions/run.tcl*

Instructions

Task 1. Load the Design

1. Change to the **`lab2_dp`** directory, invoke IC Compiler and start the GUI:

```
UNIX% cd lab2_dp
UNIX% icc_shell -gui
```

2. Open the **orca_setup** cell from the **orca_lib.mw** design library.

Note: This cell has gone through "data setup".

3. Take a look at the *LayoutWindow*. The large greenish-blue rectangles are the macro and IO pad cells, and the small purple rectangles in the lower left corner (zoom in if you want to see them more clearly), are the standard cells. All of these cells are instantiated cells in the netlist. They are all stacked on top of each other at the origin (0,0).

4. Apply timing and optimization controls which are specified in *./scripts/opt_ctrl.tcl*:

Note: Most of these settings are discussed in the *Appendix* of Unit 1. Some are discussed in later Units. Do not spend time here trying to understand them:

```
source scripts/opt_ctrl.tcl
```

5. Switch to the *Design Planning* task menu in the *LayoutWindow* by selecting: **File → Task → Design Planning**

Task 2. Initialize the Floorplan

1. The logical netlist from synthesis does not contain *physical-only* cells such as power and ground pad cells or corner pad cells. You have to therefore create these extra cells before being able to physically place them in the periphery area of your chip. Create the corner and P/G cells and define all pad cell positions using a provided script:

Hint: *When typing use the **[Tab]** key for command/option/file completion.*

```
source –echo scripts/pad_cell_cons.tcl
```

Look at the log output to verify that these cells have been created and constrained without any error or warning messages.

In a separate UNIX window look at the above script to help you answer the following questions. Check your answers against the solution in the back.

Question 1. What is the command to create a pad cell called **VDD_TEST** using the reference cell **pvdi**? (Do not run this command!)

.....

Question 2. What "side" is used to define the location of the upper-right corner cell (***cornerur***)?

.....

2. Initialize the floorplan:

Select **Floorplan→Initialize Floorplan…**
Change the *Core utilization* to **0.8** (80%).
Change the *Core to left/right/bottom/top* spacing to **30**.
Click **OK.**

3. Fit **[F]** the *LayoutWindow* and have a look at the chip's core and periphery areas. The blue hash-marked rectangles outside the chip along the top edge are the unplaced macro cells. The purple objects along the right edge are all the standard cells.

Note: The corner cells are easily visible – look at the large blue square that takes up the full layout view (labeled *pfrelr*). There are four groups of four P/G pads placed in the middle of each side.

4. Zoom into the periphery area of the chip and notice that the spacing between all the pads is about equal.

Question 3. Will the pads always be spaced equally? Explain.

.....

.....

.....

5. Insert the *pad fillers* to fill the gaps between the pads. Depending on the technology and library being used, this may be needed for N- or P-well and/or for power/ground pad ring continuity. To keep the number of pad filler cells required to a minimum, specify the larger filler cells first in the list. Otherwise, a 1,000 um space will get filled with 200 x 5 um width cells, instead of one 1,000 um width cell. Enter the command in the box below or `source` the provided script `scripts/insert_pad_filler.tcl`:

```
insert_pad_filler –cell "pfeed10000 pfeed05000 \
pfeed02000 pfeed01000 pfeed00500 pfeed00200 \
pfeed00100 pfeed00050 pfeed00010 pfeed00005"
```

6. Zoom into the space between two pad cells and notice the filler cells that have been inserted.

7. Make the "logical" connection (no physical routing) between the power/ground signals and all power/ground pins of the I/O pads, macros and standard cells, by executing the following script:

```
source –echo scripts/connect_pg.tcl
```

Note: There are 3 different power supplies in this design: *VDD*, *VDDQ* and *VDDO*. The latter two are used in the periphery of the chip.

8. Build the PAD area power supply ring:

```
create_pad_rings
```

Zoom into the area between the pads to see that metal routes have been added, over the filler cells, to connect the existing power routes within each pad cell to form continuous P/G pad rings.

9. Save the design as "floorplan_init":

```
save_mw_cel –as floorplan_init
```

Task 3. Preplace the Macros Connected to I/O Pads

In this task you will identify the macros that are connected to I/O pad cells and you will manually place them in the core area such that their connections to the I/O pads are as short as possible.

1. Zoom in to see the top periphery area shown in **Figure 1** below.

2. Identify macros that connect to I/O pads, as follows:

   Choose **Select → Cells → By Types…**
   Click the **Uncheck All** button in the top (*Cell Type*) section of the dialog box.
   Select the **Macro** cell type check box.
   Click the **Select All** button in lower left corner of the dialog box.
   Click **OK**.
   Notice that all the macros are now selected – highlighted in white.

   Select the **Flylines** button  from the top banner section of the *LayoutWindow*.

   In the "Show flylines" panel that appears on the right side of the window click on the pull-down menu and choose **Selected to IO** and **Apply**.

Reduce the “brightness” to 50% or less to better see the three flylines. The three lower left macros show connections to the top IO pads.

3. Keep the “Show flylines” panel open and, if needed, adjust the viewing area (pan/zoom) to see the picture below. If you accidentally unselect the macros and the flylines disappear, use the **[Ctrl]** key to re-select the three circled macros shown here, and the flylines will re-appear.

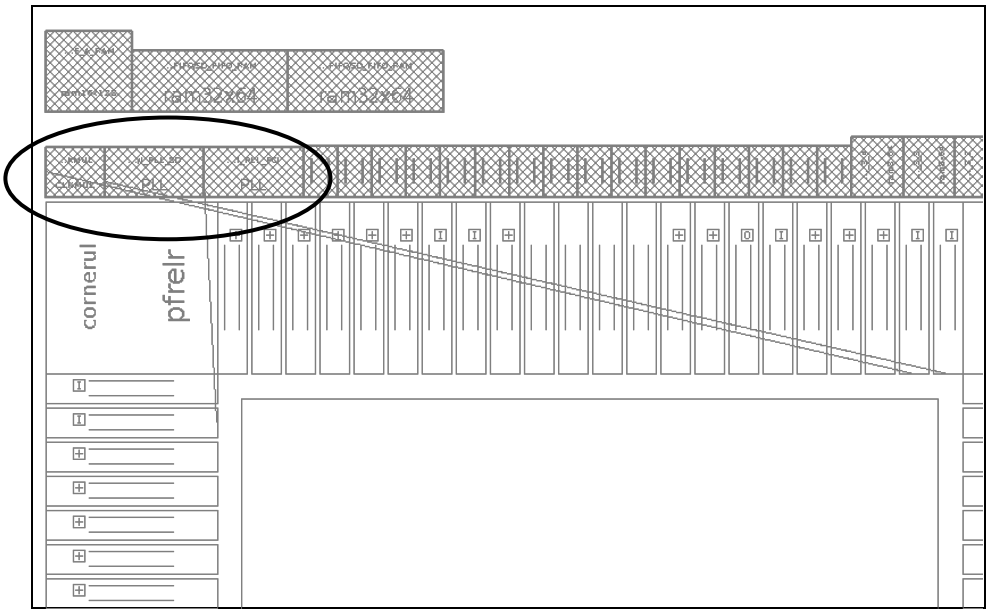


Figure 1. Flylines of the IO ←→ macro connections.

4. The three macros with a direct connection to IO pad cells are called *I_CLOCK_GEN/I_PLL_PCI*, *I_CLOCK_GEN/I_PLL_SD* and *I_CLOCK_GEN/I_CLKMUL*. Hover your mouse arrow over a cell to see its information window in the lower-left area. The two *PLLs* in this design should be placed towards the top left and right corners of the chip so they are closer to their respective clock pads.

Question 4. Which IO pad cells are these *PLLs* connected to?

.....

.....

5. Now you will manually move the **I_PLL_PCI** macro, which is connected to the left pad, into the core area. Keep in mind that you can use the undo button to back track your steps.

a. Select just the **I_PLL_PCI** macro using **Selection Tool** button.

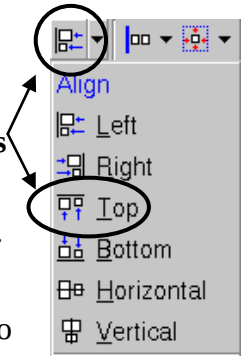
b. Select the **Move/Resize Tool (M)** button (may be in the left banner of the window) to begin the moving process.

Drag the **I_PLL_PCI** macro to approximately the top-left corner of the core area. Leave some room to the edges of the core.

c. With the PLL still selected, use the align functions to align the PLL to the top and left edges of the core:

Click the **Align Objects to Left** button to align it to the core's left edge

From the pull-down menu select the **Align Objects to Top** button to align it to the core's top edge



The PLL is now aligned with the edges of the core.

d. To make sure that the cell is not moved by virtual flat placement, click on the "padlock" button to lock it down. You should see an X through the cell now. This is an alternative to using `set_dont_touch_placement`.

6. Try to move the "fixed" I_PLL_PCI macro. You should not be able to do so. If you are able to move it, use the undo button to put it back and "fix" it in place. Don't worry if you make a mistake since you will be provided with a script to place these macros at the expected coordinates in a later step.

Use the **[ESC]** key as needed to return the cursor to the "select" mode.

7. In the next steps you will repeat the steps above to move the other two macros into the core area and near their respective IO pad cells. DO NOT spend too much time on this step to get them perfectly placed. A script in the next step will ensure correct placement:

Click on **I_PLL_SD** to select it.

Select  and drag it to the top-right corner of the core area.
Align it to the top and right edges.
Rotate 180^{o}  → 

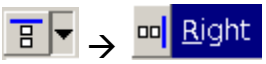
to reduce its wirelength.
Click the "padlock" button to lock it down.

Select and drag **I_CLKMUL** to the left side of **I_PLL_SD**.
Align it to the top edge.
From the"Rotate" pull-down menu select **Y-axis** to mirror along the Y-direction (= flip in the X-direction) to reduce the wirelength.

To space **I_CLKMUL** 10 microns from **I_PLL_SD**:
Hold down the Ctrl key and select both **I_CLKMUL** and **I_PLL_SD.**

Specify a *distribute offset* of **10** 

.

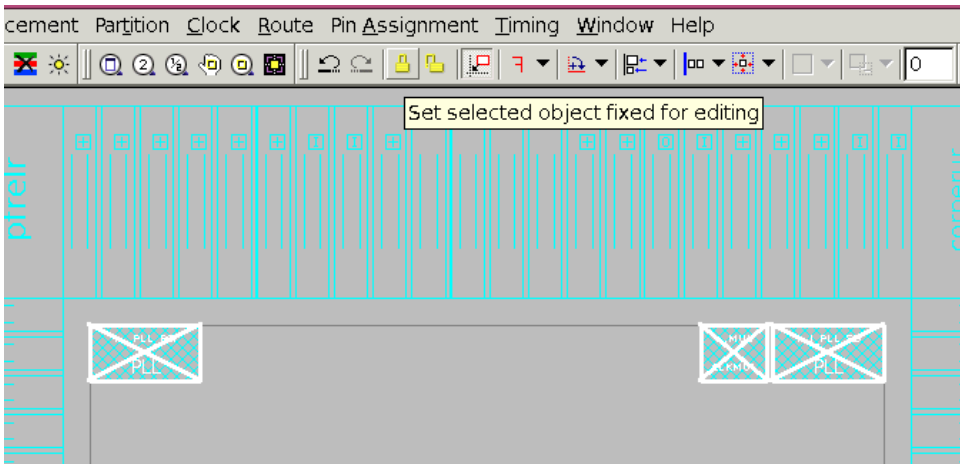
Select the **Distribute Objects to Right** button 

.

Lock down the **I_CLKMUL** macro.

Close the *flylines panel* on the right by clicking on the small "x" 

.



You should see an "X" in each of the three macros when they are selected. The remaining macros will be placed concurrently with standard cells during VF placement at a later step.

8. To ensure that the three macros are placed as expected, you can source the following script:

```
source -echo scripts/preplace_macros.tcl
```

Task 4. Perform Virtual Flat Placement

Normally, prior to performing virtual flat placement, any known *macro placement constraints* as well as *hard/soft placement blockages* should be defined. We will skip these steps initially so that you can observe the "default" VF placement behavior. After the first placement the above constraint will be applied and VF placement will be performed again. You will notice a marked difference.

1. Verify that the current VF placement strategy options have default settings:

```
report_fp_placement_strategy
```

Question 5. What is the current default *sliver size* distance?

.....

2. Apply a *sliver size* of **10** to prevent standard cells from being placed in narrow channels (< 10 um) between macros:

```
set_fp_placement_strategy -sliver_size 10
```

3. Execute a timing-driven VF placement with "no hierarchy gravity" (to ensure that the "logical hierarchy" does not affect placement of this non-hierarchical or flat layout):

```
create_fp_placement –timing_driven \
   –no_hierarchy_gravity
```

4. Examine the global route congestion map:

Click on the *Global Route Congestion* button  .
Click the **Reload** button on the pop-up panel.

A dialog box appears which contains the command to be executed for congestion analysis:

```
report_congestion -grc_based -by_layer \
                  -routing_stage global
```

Select "**OK**".

5. An "**Errors**" box appears in the GUI – click **OK**.

If you scroll up in the log to the beginning of the congestion analysis output, you will see an "Error" message (PSYN-348) about a macro that "is not fixed". You can safely ignore this. You could "fix" all macro placements prior to the congestion analysis to avoid the PSYN-348 error. However, you will be running `create_fp_placement` again to modify macro placement, so you don't want fixed placement on the macros yet.

You should not have any congestion issues. There are overflow GRCs but they are scattered over the core area.

6. Close the *Global Route Congestion* panel on the right by clicking on the small "**x**" in its upper right corner.

7. Routing of power and ground straps and macro rings for this design can be made easier if we turn some of the macros into arrays. Source the script below to apply macro placement constraints to accomplish the following goals:

- Place macros as close to the edges of the chip as possible
- Group macros together as much as possible
- Turn on *virtual IPO* to mimic timing optimization (and prevent unnecessary placement optimization)
- Limit the legal placement orientation of some RAMs

```
source –echo scripts/macro_place_cons.tcl
```

8. Double check your settings. Suggestion: *Use the up-arrow in the icc_shell window to find and re-execute the "strategy" command*:

```
report_fp_placement_strategy
report_fp_macro_options
```

9. Source the following script to set a *hard keepout margin* of **10** microns around all macros. This will make it easier to create P/G rings around the macros and avoid congestion as well as signal routing DRCs around the macros:

```
source –echo scripts/keepout.tcl
```

10. Take one last look at the macro placement before running the VF placer again:

```
create_fp_placement –timing_driven \
   –no_hierarchy_gravity
```

Note: Notice that the macro placement is very different – a lot of 'grouping' of similar macros, except for the manually placed ones, which are "fixed" in place.

11. Analyze the global route congestion map again. You need to click **Reload → OK** to update the map. An "Error" box appears in the GUI - click **OK** (similar to step5).

There should not be any congestion issues.

Close the analysis panel on the right by clicking on the small "**x**".

12. Lock down all macros:

```
set_dont_touch_placement [all_macro_cells]
```

13. Save the cell:

```
save_mw_cel –as floorplan_placed
```

Task 5. Create P/G Rings Around Macro Groups

In the task following this one you will use "Power Network Synthesis" (PNS) to automate the creation of power/ground core and individual macro rings, as well as vertical and horizontal straps. If you want to create rings around groups of macros, that is done prior to PNS, which is what this task will accomplish.

1. We have created a script to create P/G rings around six groups of macros. Take a look at the file located at *./scripts/macro_pg_rings.tcl*. The P/G rings are created by:
- Defining a rough "region" that encompasses a group of macros
- Defining the block ring layers, widths and offsets
- Creating (committing) the metal routes

2. Execute the script:

```
source ./scripts/macro_pg_rings.tcl
```

3. Take a look at the rings that have been created.

Notice that the "`PLL`" macro in the upper-left corner is the only macro that does not have a P/G ring around it – this will be done by PNS.

Notice also that, in addition to the rings around the macro groups, there are vertical/horizontal straps in between the macros. This is nice feature of the `create_fp_group_block_ring` command. It can be disabled with the `–skip_strap` option, if preferred.

Task 6. Power Network Synthesis

The power "grid" needs to be completed. You could create P/G straps that feed the center of the core, a core ring, as well as rings around individual macros "manually" (similar to the way the macro group rings were created in the previous task), but to do so would require you to guess the appropriate number and width of the straps, as well as the width of the core ring to achieve acceptably low IR drop. Instead, you will use IC Compiler's *Power Network Synthesis* (PNS) capability to automatically determine the number and width of straps, as well as the core ring width, based on a target IR drop. You can experiment with different goals, and when acceptable results are achieved you then "commit" or physically implement the power grid.

If you are running short on time, you can source the `scripts/pns.tcl` to perform steps 1-5 below. Skip steps 6-7 if you use this script.

First, you will use the **Preroute→Power Network Constraints** menu to apply a number of constraints for the core rings, the macro rings as well as the vertical and horizontal straps:

1. Apply the strap constraints:
Preroute→Power Network Constraints→Strap Layers Constraints...

Note: If you don't see this menu switch the *task* menu to Design Planning: **File →Task → Design Planning**

Select the **METAL5** *Layer* and set the *Direction* to **Horizontal**.

Set the "By strap number" *Max* to **24** and *Min* to **2**.

Set the *Width Max* to **4** and the *Min* to **2**.

Set the *PG Spacing* to **Microns** and enter **0.6**.

Click the **Set** button.

Repeat the steps for **METAL4** with *direction* **Vertical,** with the same min/max number of straps, min/max widths, and spacing.

Set then **Close** the dialog.

2. Apply the core ring constraint:
Preroute→Power Network Constraints→Ring Constraints...

Select **METAL3** (Horizontal) and **METAL2** (Vertical) for the core ring.

Select the **Ring width** option and choose **Variable**.

Set the *Max* to **12** and the *Min* to **10**.

Click the **Set** button.
Close the dialog.

3. Define a macro ring for the PLL macro without a ring:
Preroute→Power Network Constraints→ Block Rings Constraints…

Select the **Specified cell instances** option.

In the *LayoutWindow* select the upper-left PLL macro.

Back in the *Block Rings Power Network Constraints* dialog, next to the selected *Specified cell instances* radial, click the  *"Sets or appends selected cells to the edit field"* button to enter the selected macro cell name in the field.

For *Power Ground nets* enter **VDD VSS**.
Set the vertical and horizontal layers to **METAL4** and **METAL5** respectively, and change the **width** to **3** for both.
Click on **Set**, then **Close**.

4. Apply global constraints:

Preroute→Power Network Constraints→Global Constraints...

Keep the existing options selected.

Select the option "**No routing over hard macros**".
Click on **Set**, then **Close**.

5. Invoke *PNS* as follows:

Open the *PNS* dialog using **Preroute→Synthesize Power Network…**

In the *Synthesize power network by nets* field enter: **VDD VSS**.

Change the **Supply voltage (V)** to **1.32** .
(The nominal voltage is 1.2 V; Use the maximum voltage of 1.32V)

Leave the *Target IR Drop* at **10% of supply voltage**.

Change the **Power budget (mW)** to **350** (the power spec for this chip).

Under the **Pads info** section, select "**Specified pad masters**".
Enter **pv0i pvdi** into the adjacent field.

Press **Apply**, and after some calculations you should see an *IR drop map*.

Question 6. What are the estimated maximum IR drops for VDD and VSS? (look at the log output)

.....

.....

Question 7. How many horizontal and vertical VDD straps, and what widths, were used to achieve the above IR drops? (look at the log again)

.....

6. Play with the "*Target IR Drop*" field to see its affect on the number and width of the straps: Set it to "**Lowest**", or set the **Specified** field to 100 mV, then **Apply** again. Observe how the network changes, and along with it the IR drop.

7. When done experimenting, set the *Target IR Drop* back to **10%**, then **Apply**.

8. Build the suggested power plan *clicking on th*e **Commit** button, or by typing:

```
commit_fp_rail
```

Cancel the dialog if it is still open.

9. Zoom into your chip to see how all PG straps and rings were created.

Notice that there are no connections between the macros and the surrounding power rings.

Notice also that there are no P/G rails along the standard cell placement rows.

10. To complete power plan we need to hook up the power pins on all macros, and create the standard cell power rails. Execute the following commands to accomplish this:

```
preroute_instances
preroute_standard_cells –fill_empty_rows \
    -remove_floating_pieces
```

Note: The WARNING about "`floating rail segments removed`" is expected because we used the "`–remove_floating_pieces`" option.

11. Zoom in and notice the macro pin connections and the blue P/G rails on metal1 (called **METAL**).

12. Now analyze the completed power plan using **Preroute→Analyze Power Network…**

Enter the same values for *Power Ground nets* (**VDD VSS**), *Power budget* (**350** mW), *Supply voltage* (**1.32** V), and *Specified pad masters* (**pv0i pvdi**) that were used previously for *Power Network Synthesis*, then press **OK**.

You will see another heat map.

Question 8. What are the final reported maximum VDD/VSS IR drops? (from the log)

.....

.....

Question 9. Can you explain why the final IR drops are smaller?

.....

.....

13. Close the *PNA Voltage Drop* on the right by clicking on its small "**x**".

14. Save the cell:

```
save_mw_cel –as floorplan_pns
```

Task 7. Check the Timing

Now that the power plan is done, you have to perform a few more steps to complete the placement and to verify max-delay (setup) timing.

1. If you are not able to see the standard cells in the *LayoutWindow*, go to the "Visibility" panel in the left margin of the LayoutWindow, expand the "**Cell**" listing by selecting the "+" sign, and make sure that "**Standard**" is checked.

2. *PNS* created many straps on METAL4 and METAL5, which were placed over the standard cells. It can be advantageous to prevent standard cell placement under the straps – this reduces the likelihood of congestion along the straps, and reduces crosstalk effects on the power nets. Apply a "complete" power net (*pnet*) blockage on the straps, then run the virtual flat placement again to take pnet settings into account:

```
set_pnet_options -complete "METAL4 METAL5"
create_fp_placement –timing_driven \
   –no_hierarchy_gravity
```

Verify that there are no longer any standard cells under the straps.

3. Since we are about to check timing, perform actual global routing by running the following command:

```
route_zrt_global
```

4. Bring up the global route congestion map (no need to "Reload). There should not be any congestion issues. Close the panel (click on small "**x**").

5. Generate a maximum-delay (setup) timing report using the "view" procedure (it will take a few seconds to update the timing and generate the report):

```
v report_timing
```

Use the search mechanism to highlight or tag the word "slack":
RE Search → type in "**`slack`**" → ***Tag***. Scroll up/down. You should see the words **`slack (MET)`** followed by a *positive* number at the end of each of the 8 clock group paths. This design meets setup timing. Click on ***Close Search*** then ***Close Window***.

Question 10. Can you analyze timing without first performing an actual global route? If so, should you?

.....

6. To fix any timing violations (and design rule violations), if there were any, you would invoke the following command and repeat global route. Feel free to do so, if you have the time, otherwise skip to the "Save the cell" step:

```
optimize_fp_timing –fix_design_rule
```

Repeat global routing, congestion analysis and timing analysis one last time. The design should not have any congestion issues or timing violations.

7. Save the cell as **floorplan_complete**.

Task 8. Write Out the DEF Floorplan File

1. Remove all the placed standard cells then write out the floorplan file in *DEF* format. The *DEF* floorplan file will be used by *Design Compiler Topographical* to re-synthesize the design using the floorplan you just designed, and will again be used by *IC Compiler* to re-create the floorplan when reading in the re-synthesized netlist (next Task):

```
remove_placement -object_type standard_cell

write_def –version 5.6 –placed –all_vias –blockages \
      -routed_nets –rows_tracks_gcells –specialnets \
      –output design_data/ORCA.def
```

2. Verify that the DEF file has been created in the *design_data* directory.

3. Close the design library without saving the design in memory:
File → Close Library → Discard All

Task 9. Create 2nd Pass Design Ready for Placement

We will now pretend that this design was re-synthesized from RTL code using Design Compiler *Topographical mode*, along with the floorplan description captured in the DEF file generated in the previous task. You have been given a 2nd pass netlist, ORCA_2.v, along with an updated constraints file, ORCA_2.sdc.

1. Perform data setup using the new ORCA netlist and constraints:

```
source scripts/2nd_pass_setup.tcl
```

This script executes the following standard data setup steps:

```
create_mw_lib orca_lib_2.mw -technology $tech_file \
   -mw_reference_library $mw_ref_libs –open

import_designs design_data/ORCA_2.v \
   -format verilog -cel ORCA -top ORCA

set_tlu_plus_files -max_tluplus $tlup_max \
                   -min_tluplus $tlup_min \
                   -tech2itf_map  $tlup_map

source scripts/connect_pg.tcl
read_sdc design_data/ORCA_2.sdc
read_def design_data/ORCA_2.scandef
source scripts/opt_ctrl.tcl
```

2. Read the DEF file that was written out in the previous task:

```
read_def design_data/ORCA.def
```

Note: You should now see the same floorplan that you designed in the previous tasks.

3. Re-apply the *pnet options* that you applied after *Power Network Synthesis* in Task 6, step 1. These settings are not captured in the *DEF* file:

```
set_pnet_options -complete "METAL4 METAL5"
```

4. Save the cell as **ready_for_placement**.

5. Exit IC Compiler.

You have completed the Design Planning lab.

Answers / Solutions

Question 1. What is the command to create a pad cell called **VDD_TEST** using the reference cell **pvdi**?

```
create_cell {VDD_TEST} pvdi
```

Question 2. What "side" is used to define the location of the upper-right corner cell (*cornerur*)?

The *upper right* corner is defined as "**-side 2**".

Question 3. Will the pads always be spaced equally? Explain.

No. If some pads defined in the *tdf* file contain an "offset" value then the offset (with respect to the bottom or left core edge) is met first before the remaining pads are spread equally in the remaining space. If no offsets are used then all the pads will be spread out as close to equal as possible.

Question 4. Which IO pad cells are these *PLLs* connected to?

The *PLLs* connect to the 2nd pad cell from the top, on the left side (*pclk_iopad*), and the right most on the top (*sdr_clk_iopad*).

Question 5. What is the current default *sliver size* distance?

0.00

Question 6. What are the estimated maximum IR drops for VDD and VSS?

From the log (*Your numbers may vary slightly*):

```
Target IR drop :  132.00 mV
Net name : VDD
IR drop of the synthesized net :  127.32 mV
...
...
Target IR drop :  132.00 mV
Net name : VSS
IR drop of the synthesized net :  127.85 mV
```

Note: The strap calculations are based on meeting the IR drop constraint for VDD. By default VSS inherits the same number and width of straps as VDD. This is why it is possible for the VSS IR drop to be different than VDD, and possibly even violate the "target" (not in this case).

Question 7. How many horizontal and vertical VDD straps, and what widths, were used to achieve the above IR drops?

From the log: (*Your numbers may vary slightly*)

```
Net name : VDD
...
Layer: metal5, Direction: Horizontal, # of Straps: 10..
The maximum width of straps: 3.000 microns
The average width of straps: 2.501 microns
Layer: metal4, Direction: Vertical, # of Straps: 10..
The maximum width of straps: 3.000 microns
The average width of straps: 2.506 microns
```

Question 8. What are the final reported maximum VDD/VSS IR drops?

From the log: (*Your numbers may vary slightly*)

```
Net Name: VSS
...
Maximum IR drop:  74.218 mV
...
Net Name: VDD
...
Maximum IR drop:  71.358 mV
```

Question 9. Can you explain why the final IR drops are smaller?

This is primarily because of the standard cell rails, which were not present during PNS. These help to further distribute power and reduce the overall power network "resistance".

Question 10. Can you analyze timing without first performing an actual global route? If so, should you?

You can, but without the global route, timing analysis will be based on "virtual routes" – orthogonal route estimates without specific layer or congestion information. Virtual routes are less accurate and correlate less well with the detailed routes to be performed later.

This page is left blank intentionally.

3 Placement

Learning Objectives

During this lab you will perform perform scan-chain aware placement and optimization for timing, area and power.

After completing this lab, you should be able to:

- Perform setup for placement and timing optimization
- Perform setup for DFT and power optimization
- Perform standard cell placement
- Analyze congestion, timing and power
- Perform incremental placement and optimization

Lab Duration: 60 minutes

Introduction

In this lab you are provided with a floorplanned design called *ORCA_floorplanned* which is ready for the placement phase. You will execute the appropriate pre-placement setup steps and then perform standard cell placement. After placement you will analyze the results and execute additional appropriate steps to improve the results.

Answers / Solutions

There is an *ANSWERS / SOLUTIONS* section at the back of each lab. You are **encouraged** to refer often to this section to verify your answers, or to obtain help with the execution of some steps.

Relevant Files and Directories

All files for this lab are located in the ***lab3_placement*** directory under your home directory.

lab3_placement/

orca_lib.mw/CEL

ORCA_floorplanned	The ORCA design after design planning, saved in *Milkyway* format

scripts/

inputs_toggle_rate.tcl	A script used to set statistical toggle rates for the input ports.
ndr.tcl	A script containing commands to define non-default clock routing rules.
opt_ctrl.tcl	Timing and optimization controls

design_data/

ORCA_TOP.scandef	Scan chain information used during scan-chain re-ordering.

.synopsys_dc.setup	Setup variables.

Instructions

Task 1. Pre-placement Settings and Checks

1. Invoke the IC Compiler "GUI" from the **lab3_placement** directory.

```
UNIX$ cd lab3_placement
UNIX$ icc_shell -gui
```

2. Open the design library, open the *ORCA_floorplanned* design cell:

```
open_mw_lib orca_lib.mw
open_mw_cel ORCA_floorplanned
```

Note: In the previous *Design Planning* lab a larger, more complex full-chip version of ORCA was used in order to illustrate IC Compiler's design planning capabilities and features. From this lab on, we will be using a simpler block-level version of ORCA. This block-level design will enable us to explore the key steps of placement, CTS and routing while keeping run times reasonable.

3. Apply timing and optimization controls:

```
source scripts/opt_ctrl.tcl
```

4. Macro placement is usually defined and "fixed" during design planning. It is, however, possible that a last minute change is made to the macro placement while forgetting to "fix" the new location. It is therefore a good idea to repeat the "fixing" step prior to placement to ensure that no macros are moved during the placement phase:

```
set_dont_touch_placement [all_macro_cells]
```

5. Verify that all process metal layers are available for routing - there should be no *ignored layers*:

```
report_ignored_layers
```

6. Verify that standard cells are allowed to be placed under the *METAL2 – METAL4* power nets, as long as no DRC violations occur (*partial blockage*):

```
report_pnet_options
```

7. During design planning both *soft* and *hard* placement *keepouts* were applied. Verify that these variables are still set and are not the default value of zero:

```
printvar physopt_hard_keepout_distance
printvar placer_soft_keepout_channel_width
```

Question 1. Since the above are variable settings, which are not saved with a design cell, how did these variables retain their non-default values?

.....

.....

Question 2. How will these variables affect placement?

.....

.....

.....

.....

8. The clock nets will be constrained to be routed on *METAL3 – METAL6*, with double-spacing rules. Non-default routing (*NDR*) rules affect congestion, which can affect placement. Apply the non-default routing rules for all clock nets, as shown below, by sourcing the ***ndr.tcl*** file provided to you:

```
# Source this file: scripts/ndr.tcl
define_routing_rule 2X_SPACING -spacings {METAL2 0.6 \
         METAL3 0.6 METAL4 0.8 METAL5 1.2 METAL6 1.4}
set_clock_tree_options -clock_tree [all_clocks] \
   -routing_rule 2X_SPACING -layer_list "METAL3 METAL6"
```

9. Verify that the *floorplanned* design is ready for placement:

```
check_physical_design –stage pre_place_opt
```

Note: There should be no errors or warnings.

10. Execute a different pre-placement check:

```
check_physical_constraints
```

Note: The messages about “Narrow Placement Area” can be ignored for this design.

11. This design includes scan chains, which were inserted during synthesis. The netlist that was read into IC Compiler during the data setup phase was a *Verilog* netlist.

Question 3. Does the *SCANDEF* information transfer into IC Compiler through the *Verilog* netlist?

.....

12. Execute the following command to confirm that no scan chain information exists:

```
report_scan_chain
```

The report is empty, which means that no scan chain annotation exists.

13. Load the *SCANDEF* file (which was generated during synthesis, after scan insertion):

```
read_def design_data/ORCA_TOP.scandef
```

Question 4. How many scan chains exist in the design?

.....

14. Generate another scan chain report. Use the “view” TCL procedure:

```
v report_scan_chain
```

You should see scan chains now! This information will be used during `place_opt` to optimize the scan chain ordering.

Question 5. What `place_opt` option is required to perform scan chain re-ordering during placement?

.....

Close the *report_scan_chain* view window: **Close Window**

Next we are going to setup for power optimization.

15. We do not have a simulator-generated *SAIF* file, which is preferred, so instead we will read in a user-generated *toggle-rate* file.

```
report_saif
source scripts/inputs_toggle_rate.tcl
report_saif
```

Notice that the toggle-rate coverage is reported under the "*User Annotated*" column.

16. We do not have *multi-V_{th}* libraries, so there is no need to modify the logical and physical library settings. Even without the multi-V_{th} libraries IC Compiler can still achieve *some* leakage power reduction by cell down-sizing and buffer removal. Leakage power optimization is enabled by default. You can confirm this with:

```
report_power_options
```

17. Enable low-power placement (LPP) dynamic power optimization. Gate-level dynamic power optimization (GLPO) will be enabled after `place_opt`, just before `psynopt`:

```
set_power_options –low_power_placement true
report_power_options
```

Note: There is a warning with set_power_options –low_power_placement command. It will be replaced by another command in the future release.

Question 6. What `place_opt` option is required to perform power optimization during placement?

.....

18. Save the current design with its pre-placement settings:

```
save_mw_cel -as ORCA_preplace_setup
```

Task 2. Placement and Optimization

1. Invoke placement and optimization using the appropriate options - congestion was not an issue during design planning:

```
place_opt –area_recovery -optimize_dft -power
```

While place_opt is running you may observe the output on the screen to get an idea of the optimizations that occur.

2. Save the current design:

```
save_mw_cel -as ORCA_place_opt
```

3. Generate a congestion map from the *LayoutWindow*:

Global Route Congestion → Reload → OK

The GUI step is the same as executing the following command:

```
report_congestion –grc_based –by_layer \
     –routing_stage global
```

The congestion level is marginal and is probably ok since they are scattered and not concentrated in a single area.

4. Close the congestion map by clicking on the small ***x*** in the *Global Route Congestion* panel.

5. Generate a physical design report and scroll to the top of the output:

```
report_design –physical
```

Question 7. What is the reported standard cell utilization?

.....

6. Generate a QoR (quality of results) report:

```
report_qor
```

Question 8. Are there any user-defined path groups?

.....

Question 9. Why are we not concerned about hold time violations?

.....

.....

7. Report the power dissipation:

```
report_power
```

Task 3. Incremental Optimization

From the previous analysis steps we have determined that we have some congestions issues and no significant timing violations. We should perform an incremental optimization to improve the design's area and power utilization. We can also enable GLPO to try to further reduce dynamic power dissipation.

1. Enable gate-level dynamic power optimization (GLPO):

```
set_power_options –dynamic true
```

Note: There is a warning with set_power_options –dynamic command. It will be replaced by another command in the future release.

2. Perform incremental logic optimization, using appropriate options:

```
psynopt -area_recovery –power
```

3. Generate a congestion map and verify that it is about the same as before:

Global Route Congestion → Reload → OK
Close the congestion map by clicking on the small ***x*** in the *Global Route Congestion* panel.

4. Generate a physical design report and scroll to the top of the output:

```
report_design –physical
```

Question 10. Has the utilization changed significantly?

.....

5. Generate a QoR report and check for any setup timing violation:

```
report_qor
```

There should not be any setup timing violation.

6. Report the power dissipation:

```
report_power
```

While there is not much improvement in power optimization since we do not have SAIF annotations and also we are not providing multi-Vt libraries. In your production design, you should expect a good improvement at this stage.

7. Save the design and exit IC Compiler:

```
save_mw_cel -as ORCA_placed
exit
```

You have completed the placement and power optimization lab.

Answers / Solutions

Question 1. Since the above are variable settings, which are not saved with a design cell, how did these variables retain their non-default values?

Since we just started a new IC Compiler session and we did not explicitly re-apply these variables ourselves, they must be in the `.synopsys_dc.setup` file, which is automatically applied whenever IC Compiler is invoked. Take a look and verify that this is indeed the case.

Question 2. How will these variables affect placement?

`physopt_hard_keepout_distance = "5"` ensures that no cells are ever placed within 5 microns of any macro's border during placement, or any subsequent optimizations.

`placer_soft_keepout_channel_width = "15"` ensures that during coarse placement no cells are placed in "channels" (the distance between any two macros, or between a macro and the core boundary) of 15 microns or less. Cells are allowed to be placed in these areas during placement legalization and subsequent optimization.

Question 3. Does the *SCANDEF* information transfer into IC Compiler through the *Verilog* netlist?

No. *SCANDEF* is only captured and automatically transferred into IC Compiler if reading in a *ddc* netlist.

Question 4. How many scan chains exist in the design?

11.

Question 5. What `place_opt` option is required to perform scan chain re-ordering during placement?

`-optimize_dft`

Question 6. What `place_opt` option is required to perform power optimization during placement?

`-power`

Question 7. What is the reported standard cell utilization?

The utilization percentage is around 81%.

Question 8. Are there any user-defined path groups?

Yes. From the path group names COMBO, INPUTS and OUTPUTS we can assume that these are user-defined I/O path groups.

Question 9. Why are we not concerned about hold time violations?

By default `place_opt` optimizes only *setup* timing violations. It ignores *hold* timing. We will address hold timing during clock tree synthesis.

Question 10. Has the utilization changed significantly?

No.

This page is left blank intentionally.

4 Clock Tree Synthesis

Learning Objectives

The purpose of this lab is for you to become familiar with the clock tree synthesis capabilities in IC Compiler.

You will perform clock tree synthesis on the ORCA_TOP design, after applying multiple settings. Reports will be generated to monitor and track the progress of the design.

After completing this lab, you should be able to:

- Set options and exceptions for CTS
- Synthesize the clock trees
- Generate and analyze clock tree skew and timing reports in order to determine CTS QoR
- Fix hold time violations and optimize the design for better area
- Route the clock nets using Non Default Routing Rules
- Analyze the routed clock nets

Lab Duration: 75 minutes

Introduction

During this lab, you will perform Clock Tree Synthesis with all the associated commands you learned in lecture.

Answers / Solutions

There is an *ANSWERS / SOLUTIONS* section at the back of each lab. You are **encouraged** to refer often to this section to verify your answers, or to obtain help with the execution of some steps.

Relevant Files and Directories

All files for this lab are located in the ***lab4_cts*** directory under your home directory.

lab4_cts/

orca_lib.mw/CEL

place_opt	A placed and scan inserted design saved in Synopsys Milkyway format.

scripts/

cts_setup.tcl	A script to set CTS options.
ndr.tcl	A script to specify non-default clock routing rules.
opt_ctrl.tcl	A script to specify general timing and optimization controls.

If you encounter problems, a command script is available to help you recover: ***.solutions/run.tcl.*** You can cut and paste from this file into the command line of
IC Compiler to execute a part of the sequence to catch up.

Instructions

Task 1. Copy and Load the Working CEL

1. Invoke IC Compiler from the lab4_cts directory.
2. Open the design library ***orca_lib.mw***.

```
open_mw_lib orca_lib.mw
```

3. Copy the CEL ***place_opt*** and name it as ***clock_opt***.

```
copy_mw_cel –from place_opt –to clock_opt
```

This cell is the equivalent result of `place_opt` placement and optimization performed in the previous lab.

4. Open the ***clock_opt*** CEL.

```
open_mw_cel clock_opt
```

Task 2. Examine the Clock Trees

1. Use the following commands and answer the questions:

```
report_clock –skew -attributes
```

Question 1. Which clock has the smallest uncertainty?

.....

```
report_clock_tree -summary
```

Question 2. How many end points or “sinks” does the clock `SD_DDR_CLK` have? Why?
(HINT: Execute `report_port` on the start point or “`Generated Source`” of `SD_DDR_CLK`)

.....

.....

```
view report_constraint -all
```

Question 3. Are there any setup (max_delay) timing violations?

.....

2. Start the GUI to perform some more analysis:

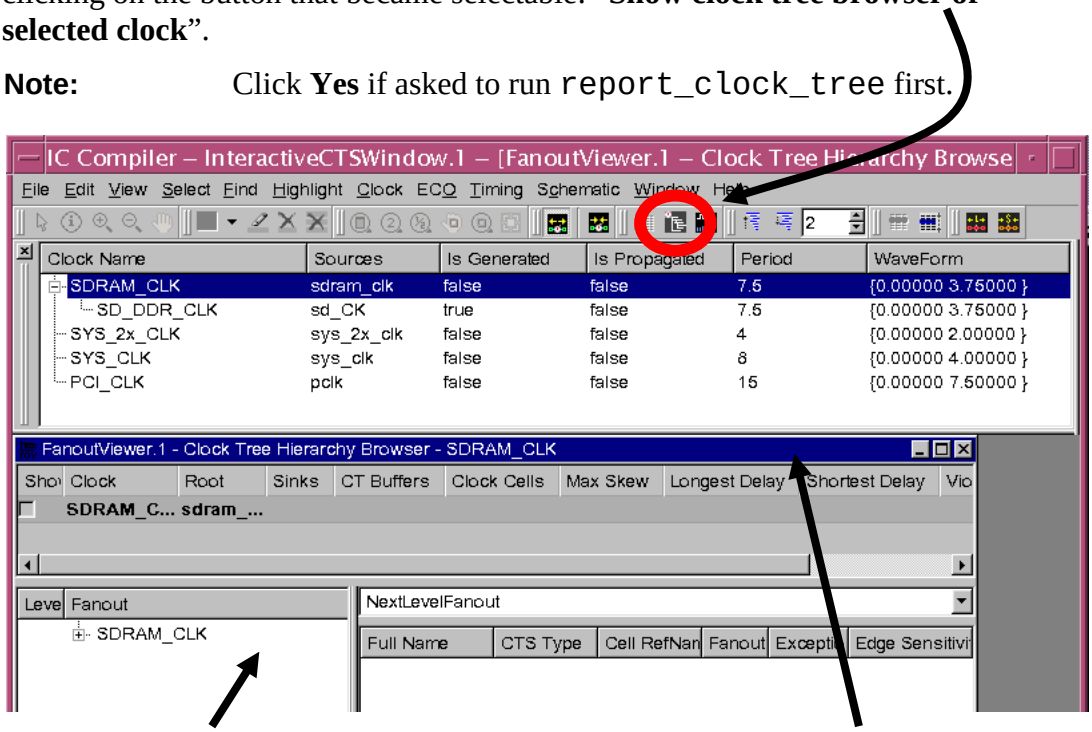
```
gui_start
```

3. Open an *InteractiveCTSWindow* by selecting **Clock→New Interactive CTS Window.**

   A new window opens listing the clocks that can be synthesized. Verify that `SD_DDR_CLK` is a generated clock (look at the `Is Generated` column) with `SDRAM_CLK` as its master clock (from the hierarchical display in the `Clock Name` column), and `sd_CK` as its source or start-point.

4. Perform closer analysis of `SDRAM_CLK` by selecting it from the list, then clicking on the button that became selectable: "**Show clock tree browser of selected clock**".

   Note: Click **Yes** if asked to run `report_clock_tree` first.



5. A new sub-window opens. Maximize it by double-clicking on its title bar.

6. Select the "+" next to `SDRAM_CLK`, then select `sdram_clk`, which is the name of the start point (an input port) of the `SDRAM_CLK` clock. On the right you should see a long list of all the sinks (end-points) connected to this clock. Use the right panel (`NextLevelFanout`) to analyze the sinks and answer the following questions:

Question 4. Are there any sinks with “exceptions”? Hint: Click on the “`Exception`” column to sort by exception.

.....

Question 5. Why are the above sinks classified as `implicit_exclude_pin`? Hint: See **Figure 1** below and look at the SDRAM_CLK sinks.

.....

.....

Task 3. Preparing for Clock Tree Synthesis

Many designs have special or non-default requirements for their clock trees, in which case executing a “default” clock tree synthesis is not sufficient.

CTS will only balance the delays (minimize skew) to “stop pins”, which, by default, are clock pins of sequential cells. If there are additional pins that need to be balanced along with these clock pins, IC Compiler needs to be explicitly told about them prior to CTS.

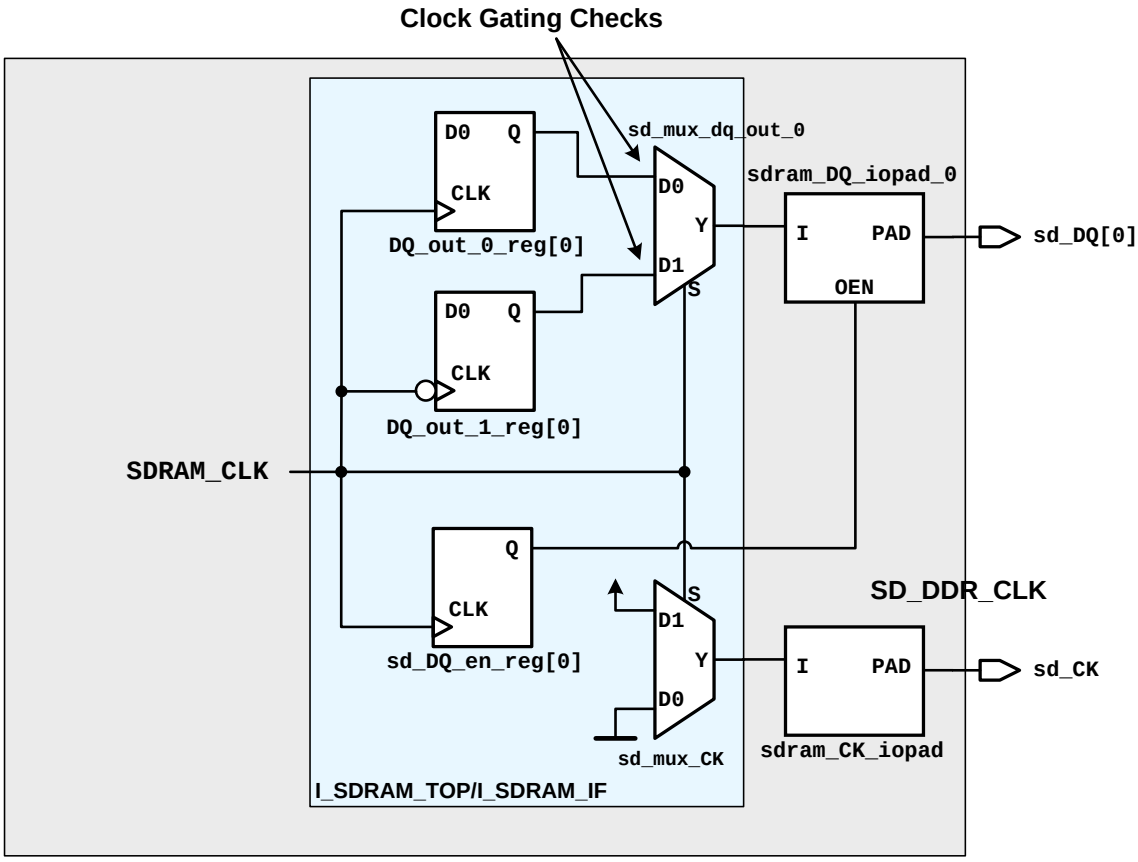


Figure 1. SDRAM interface

Figure 1 shows the *SDRAM* interface. The clock *SDRAM_CLK* is connected directly to the select pins of muxes, which in turn, will drive the output PADs at the chip level. In this lab we are using the block-level design called ORCA_TOP, so the MUX outputs are connected to output ports `sd_DQ` and `sd_CK` instead of IO pad cells. The "dummy" mux driving `sd_CK` is required because of the tight timing requirement of the DDR SDRAM interface, which produces data at its output data ports on both the rising and the falling clock edges. This design requires that the clock skew from `SDRAM_CLK` to `sd_DQ` and `sd_CK` be optimized. By default select (S) pins are marked as "implicit_exclude_pin". To have CTS balance the skew you need to redefine these select pins as "stop pins", as shown in lecture.

In the following steps, if you get stuck, use the man page and, if necessary, the ANSWERS section in the back of this lab.

1. Use the command learned in lecture to define the S pins of the muxes you discovered earlier as "stop_pin". Using the wildcard character "*" you should be able to specify all 18 exceptions using one command.

If you applied the command correctly, you should now see the GUI show the pins as "stop_pin" (click on the "Exception" bar to sort by this column).

NextLevelFanout

Full Name	CTS Type	Cell RefNa	Fanout	Exception
I_SDRAM_TOP/I_SDRAM_IF/sd_mux_dq_out_15/S	sink	mx02d4		stop_pin
I_SDRAM_TOP/I_SDRAM_IF/sd_mux_dq_out_14/S	sink	mx02d4		stop_pin
I_SDRAM_TOP/I_SDRAM_IF/sd_mux_dq_out_13/S	sink	mx02d4		stop_pin
I_SDRAM_TOP/I_SDRAM_IF/sd_mux_dq_out_12/S	sink	mx02d4		stop_pin

2. Use the GUI or the shell to set the clock target skew to 0.1 for all clocks. Relaxing the skew target from the default of 0 will speed up clock tree synthesis (`clock_opt –only_cts –no_clock_route`). For this design, a skew of 0 is not needed.

3. Adjust the clock uncertainties for post-CTS timing analysis and optimization (`clock_opt –only_psyn –no_clock_route`):

For this design, to account for clock jitter, issue commands that will set the **setup and hold uncertainties** for all clocks to **0.1**.

Note: The clock uncertainties must be adjusted *before* performing timing optimization (`-only_psyn`) to prevent the original uncertainties (which include estimated skews) from being used during this phase.

4. It is always a good idea to specify the buffers to be used for CTS. You should look for buffers in your library that have balanced rise and fall times. This will generally improve quality of the results.

Using the `set_clock_tree_references` command, specify the following buffers to be used during the "DRC buffering" phase of CTS: **`bufbd1 bufbd2 bufbd4 bufbd7 bufbdf`**.

To simplify the lab, we will not specify buffers for the skew balancing (`-sizing_only`) and delay insertion (`-delay_insertion_only`) phases of CTS.

Note: For actual designs it IS recommended to also specify buffer lists for the latter two phases of CTS!

5. Configure CTS to use non-default routing. You can either explicitly execute each of the A-B-C steps below, or you may instead execute a pre-written script (**scripts/ndr.tcl**) for all of three steps:

A. Define an NDR rule called `CLOCK_DOUBLE_SPACING` with these spacing values:

```
METAL3   0.42   METAL4   0.63    METAL5   0.82
```

B. Report the defined NDR rule and verify that the spacing values are specified correctly:

```
report_routing_rule CLOCK_DOUBLE_SPACING
```

C. Direct CTS to use `CLOCK_DOUBLE_SPACING` NDRs on all clock route segements, except for the "first level" (level 1) sinks, which should use default routing rules; Restrict the clock routes to metal layers `METAL3` – `METAL5`.

6. Verify that the "`Layers Available for Clock Routing`" and "`Buffers Available for Clock Tree Synthesis`" have been correctly defined.

```
v report_clock_tree -settings
```

7. Perform the following check prior to running CTS:

```
check_physical_design –stage pre_clock_opt -display
```

In addition to the standard output, this command generates a report in HTML, which is automatically loaded in a web browser. You are encouraged to explore the browser and follow the links to see how easy it is to obtain information generated from the report.

Note: The CTS-821 warning is reported for any generated clock that can not be traced back to its master source. During the CTS setup of this design, a “stop_pin” exception was defined on the S pin of the muxes, which breaks the clock path between the master clock source of SDRAM_CLK and its generated clock source (sd_CK). Since this is intended, you can safely ignore this warning.

Note: The CTS-832 warning is reported for SD_DDR_CLK clock defined at port sd_CK and loops to itself. This is a simple loop and CTS should run fine. In a more complex loop where sequential cell is involved, the user may have to break the loop prior to running CTS.

Quit the web browser when done.

8. Set the clock delay calculator to *Arnoldi*. This causes the delay calculation to use the more accurate Arnoldi-based delay models on clock nets in a post-CTS design (instead of the default *Elmore*-based delay model).

```
set_delay_calculation –clock_arnoldi
```

9. Use the following command to check for any clock tree issue:

```
check_clock_tree
```

The same CTS-821 and CTS-832 warnings are reported and can be ignored for this design. You are now ready to build the clock trees!

Task 4. Perform Clock Tree Synthesis

The following steps follow the “ICC QoR Reference Methodology (RM)” flow.

1. Synthesize all clock trees, without any timing optimization or routing:

```
clock_opt –only_cts –no_clock_route
```

2. Review the global skew summary after CTS:

```
report_clock_tree -summary
```

Record skew and latency for the indicated clocks:

	Skew	Longest Path
SYS_CLK		
SYS_2x_CLK		

3. Generate a different skew report using clock timing command:

```
report_clock_timing -type skew –significant_digits 3
```

Record skew and latency for the indicated clocks:

	Skew	**Longest Path (max Latency)**
SYS_CLK		
SYS_2x_CLK		

Question 6. Why are the skews reported by `report_clock_tree` and `report_clock_timing` different?

.....

.....

.....

4. Generate a timing report:

```
v report_timing
```

From `clock network delay (propagated)` you can confirm that all clocks network delays have been set to “propagated” mode automatically post-CTS. This means that for clock latency calculations, the actual clock network delays are being calculated and propagated (instead of using the estimated “ideal” values specified in the SDC constraints).

5. Generate constraint report for all violations:

```
v report_constraint -all
```

Question 7. Are there any setup or hold timing violations?

.....

6. Save the design CEL as ***clock_opt_cts***.

Task 5. Perform Hold Time Optimization

In this task you will fix the hold time violations and you will perform area recovery.

1. Enable hold time fixing:

```
set_fix_hold [all_clocks]
```

Unless the above attribute is set on the clocks, IC Compiler will not perform any hold time optimization.

At this stage you could perform hold time optimization using `psynopt`. Instead, you will perform hold time optimization in addition to area recovery. It is a good idea to perform area recovery at this stage because the clocks are now fully defined, and the critical paths are well understood.

2. Generate a QoR report:

```
report_qor
```

Note the "Design Area": _____

You could also use the `report_area` *command.*

3. It is highly recommended to use the area critical range setting (see *Appendix* of the *Data Setup* Unit 1). Area recovery is done by down-sizing, and thereby slowing down paths with positive timing slack. This down-sizing happens up to a point when the timing slack reaches 0.0 ns, by default. By applying an "area critical range" of 0.2, for example, we are preserving 0.2ns of timing margin (at the expense of a little less area recovery). This can be useful during the signal routing stage, where "crosstalk" effects may show up and further degrade our timing.

```
set_max_area 0  ;# This is already set previously
set physopt_area_critical_range 0.2
```

4. Ensure that current RC extraction data is available to capture the more accurate "integrated clock global router" (ICGR) data for clock nets (as opposed to virtual-route based data). This is a "good practice" step that should always be performed prior to `clock_opt –only_psyn ...` :

```
extract_rc
```

Note: We executed a `report_timing` in the previous task, which implicitly already performed this RC extraction. IC Compiler will not un-necessarily re-execute RC extraction if it was already performed, and if nothing has changed since then. You will notice that this step completes quickly and returns a "1". This implies that RC re-extraction was not repeated.

5. Perform post-CTS timing, area and scan chain optimization, without clock routing:

```
clock_opt –only_psyn -area_recovery –optimize_dft \
          –no_clock_route
```

During the timing optimization phase, notice the additional column titled "`MIN DELAY COST`". This indicates that hold time optimizations are now being performed as well, which is due to `set_fix_hold`.

6. Analyze the design to make sure that all violations – setup, hold, design rules – are fixed. Generate a QoR report and compare the Design Area to the number recorded above.

Question 8. Why do you think the design area has increased slightly?

.....

7. Save the cell as ***clock_opt_psyn***.

Task 6. Route the Clocks

1. Route the clocks:

```
route_zrt_group -all_clock_nets \
   -reuse_existing_global_route true
```

This will perform global routing, track assign and detail routing on all the clock nets in the design.

2. Analyze all constraints one last time to ensure there are no timing/DRC violations. Since there are no violations, there is no need to perform any additional timing optimization or clock tree optimization.

3. Save the cell as ***clock_opt_route***.

4. If the GUI is closed, open it using "gui_start" and look at the *LayoutWindow*.

5. In the **View Settings** panel turn off the visibility for **Cell**, as well as for **Power** and **Ground** under the sub-category **Net Type** of the **Route** category. You can visually see now that most of the clock nets are routed on METAL3-METAL5.

6. To obtain actual wiring statistics:

```
report_design –physical
```

Toward the bottom of this report, under the **Signal Wiring Statistics** section, you can verify that most of the wires are on METAL3-METAL5.

7. To analyze the clock net level topology, select "**Clock Trees**" visual mode to display the clock tree **type** and **level** information.

In the panel that opens, select "**Reload**", then select the clock trees you want to see in the pop-up dialog box. Try using the panel on the right to turn on/off each clock level.

Note: Level 0 is the net that connects to the root. Level 1 is the net that is driven by the first driver, etcetera.

8. Exit IC Compiler.

Congratulations! You have completed Clock Tree Synthesis lab.

Answers / Solutions

Task 1. Load the Design

Start IC Compiler, copy and load the design.

Task 2. Examine the Clock Trees

Question 1. Which clock has the smallest uncertainty?

SD_DDR_CLK with 0.05 ns.

Question 2. How many end points or "sinks" does the clock `SD_DDR_CLK` have? Why?
(HINT: Execute `report_port` on the start point or "`Generated Source`" of `SD_DDR_CLK`)

Zero. `SD_DDR_CLK` is a generated clock defined on the port `sd_CK` (as shown by `report_clock`). From `report_port sd_CK` you will see that this is an output port and it is not driving any load.

Question 3. Are there any setup (max_delay) timing violations?

There are no setup timing violation. There are many hold (min_delay) timing violations which will be addressed after the clock trees are inserted.

Question 4. Are there any sinks with "exceptions"? Hint: Click on the "`Exception`" column to sort by exception.

Yes.

Question 5. Why are the above sinks classified as `implicit_exclude_pin`? Hint: See **Figure 1** below and look at the SDRAM_CLK sinks.

There are 18 pins that are classified as "implicit_ exclude_pin". This is because these pins are SELECT pins of MUXes. You can expand the column header in the browser to see the pin names. These pins will be ignored for clock tree skew and latency optimization.

Task 3. Preparing for Clock Tree Synthesis

```
# Step 1
set_clock_tree_exceptions \
    -stop_pins {I_SDRAM_TOP/I_SDRAM_IF/sd_mux_*/S}

# Step 2
set_clock_tree_options -target_skew 0.1

# Step 3
set_clock_uncertainty 0.1 [all_clocks]

# Step 4
set_clock_tree_references -references \
      {bufbd1 bufbd2 bufbd4 bufbd7 bufbdf}

# Step 5
define_routing_rule CLOCK_DOUBLE_SPACING \
    -spacings {METAL3 0.42 METAL4 0.63 METAL5 0.82}
report_routing_rule CLOCK_DOUBLE_SPACING
set_clock_tree_options \
   -routing_rule CLOCK_DOUBLE_SPACING \
   -layer_list {METAL3 METAL5} \
   -use_default_routing_for_sinks 1
```

Task 4. Perform Clock Tree Synthesis

Question 6. Why are the skews reported by `report_clock_tree` and `report_clock_timing` different?

The `report_clock_tree` command reports “global” skew, meaning the maximum skew across the entire clock domain (the difference between the longest and the shortest insertion delays), even if these extreme clock paths are not related. Furthermore, `report_clock_tree` does not take into account any timing derating (`set_timing_derate`). This report is useful to understand what CTS does, because CTS is based on “global skew”.
The second report calculates actual (or “local”) clock skew: The reported maximum skew is between two flip flops of that clock domain that share a timing path (one clock branch launches the data while the other branch captures the data). The `report_clock_timing` command also considers `set_timing_derate`. The latter report is more accurate in determining the real worst-case clock skew.

If no timing derate is applied, `report_clock_tree` will generally show larger skews than `report_clock_timing`. In our design, however, since we have applied timing derating (confirm with `report_timing_derate`), the `report_clock_tree` skews appear to be smaller (since it does not take timing derating into account).

Question 7. Are there any setup or hold timing violations?

There are hold violations. You can ignore the max area violation since the constraint for “max area” is set to zero.

Question 8. Why do you think the design area has increased slightly?

Hold time fixing added many buffers, which increase the design area. The area would most likely have been larger without “area-recovery”.

This page is left blank intentionally.

5 Routing

Learning Objectives

After completing this lab, you should be able to:

- Perform routeability checks on a placed design with clock trees
- Apply routing options
- Perform initial route and post-initial route optimization
- Analyze the design for timing, logical and physical DRC, and LVS violations
- Fix LVS errors
- Use the color highlighting facility to analyze various aspects of the design

Lab Duration: 60 minutes

Introduction

The purpose of this lab is to familiarize you with the routing capabilities in IC Compiler. You will open a cell that has gone through clock tree synthesis, and will execute the necessary commands to route the design. We will be using the *Zroute* router for this lab.

Answers / Solutions

There is an *ANSWERS / SOLUTIONS* section at the back of this lab. You are **encouraged** to refer often to this section to verify your answers, or to obtain help with the execution of some steps.

Relevant Files and Directories

All files for this lab are located in the ***lab5_route*** directory under your home directory.

`lab5_route/`

`orca_lib.mw/CEL/clock_opt_route`	The *ORCA_TOP* design, after CTS and clock routing – the starting cell for this lab.
`./scripts`	Provided scripts for use in this lab.

If you encounter problems, a command script is available to help you recover: ***./.solutions/run.tcl***

Instructions

Task 1. Load the Design and Common Settings

1. Change to the `lab5_route` directory, then invoke IC Compiler. Make a working copy of the *CEL* named `clock_opt_route` from the library `orca_lib.mw`, then open the copy:

```
open_mw_lib orca_lib.mw
copy_mw_cel -from clock_opt_route -to signal_route
open_mw_cel signal_route
start_gui
```

This cell is the result of clock tree synthesis and clock tree routing from the CTS lab.

2. Post-CTS we want to use slow or "maximum" delays for setup timing checks and optimization, and fast or "minimum" delays for hold timing. The library variable settings are not stored with the CEL. We have specified these variables in the `.synopsys_dc.setup` file. Check the library listing to verify that each loaded "max" library has a corresponding "min" library:

```
list_libs
```

3. Load the *common settings* which were used during the placement and CTS phases of this design, and which are also required for the routing phase:

```
source scripts/common_optimization_settings_icc.tcl
source scripts/common_placement_settings.tcl
source scripts/common_post_cts_timing_settings.tcl
source scripts/common_route_si_settings_zrt_icc.tcl
```

Note: If you look at the contents of the above files, you may notice that many commands are "commented out". These files come from Synopsys' "Reference Methodology" scripts, which you can download from `https://solvnet.synopsys.com/rmgen`. The commands that have been commented out are intended to be used as needed, for non-default situations.

Task 2. Ensure that the Design is Ready for Routing

We have purposely introduced a pre-route problem in this design, to enhance the learning experience. You will uncover and fix the pre-route problem in this task.

1. Analyze the design for setup and hold timing, as well as logical DRCs:

```
report_constraint -all
```

Question 1. Are there any timing or logical DRC violations?

.....

2. Verify that there are no *ideal nets* and no *high fanout nets*:

```
all_ideal_nets
all_high_fanout -nets -threshold 501
```

If the commands return nothing then the design doesn't have *ideal nets* or nets with a *fanout* that is greater than 500. You can rerun the *all_high_fanout* command with lower threshold values if your HFN (high fanout net) strategy allows unbuffered HFNs at lower levels. It is important to verify that the design is free of unbuffered HFNs at this point. You would want to create a buffer tree for any remaining HFN that needs one, prior to routing the design.

3. Verify that the *preferred routing directions* are as expected, and that *TLUPlus* files are loaded:

```
report_preferred_routing_direction
report_tlu_plus_files
```

4. Verify that all placements are legal:

```
check_legality
```

5. Verify that all power and ground pins are physically connected to P/G nets:

```
verify_pg_nets
```

Question 2. Are there any P/G issues?

.....

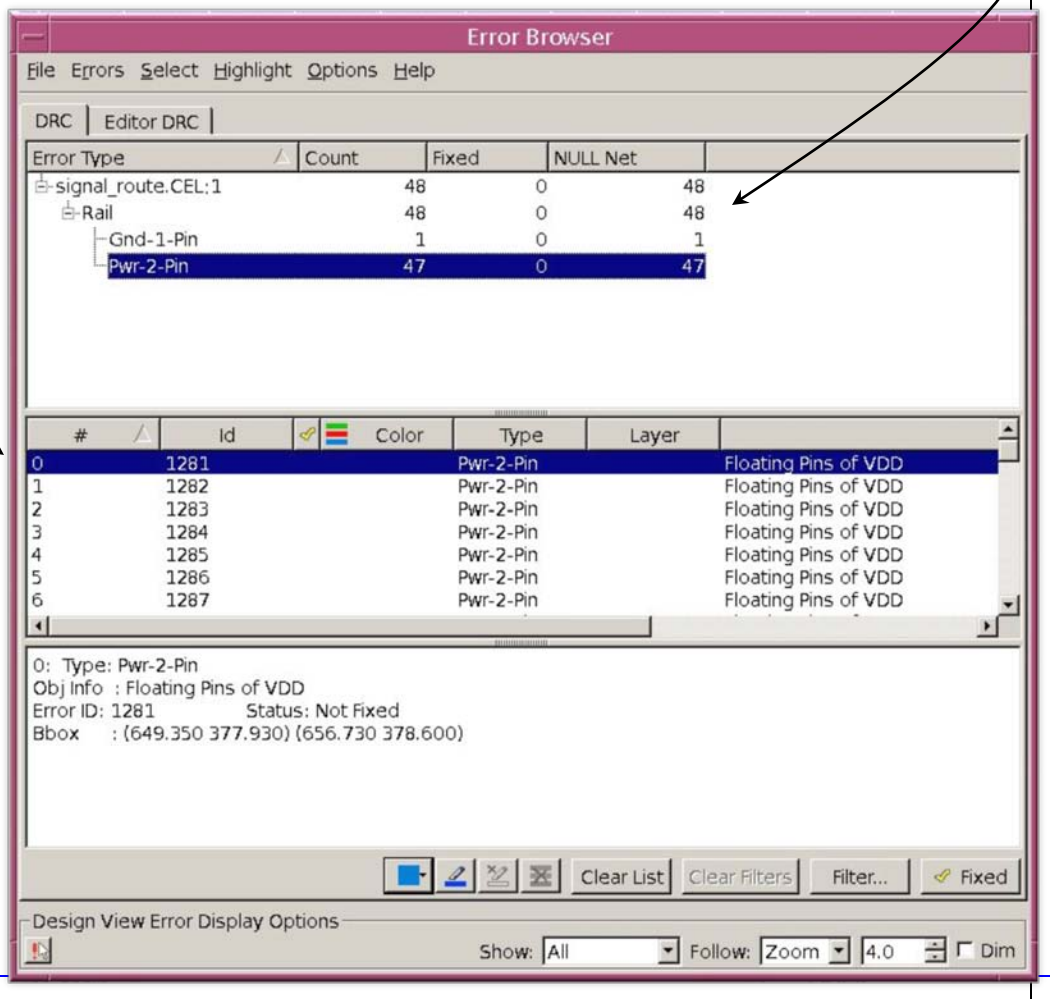
6. We will use the *error browser* to locate the problem:

In the *LayoutWindow* select **Verification → Error Browser …** to open the "Error Browser" dialog window.
Check the "**Rail**" box (since this is a P/G problem).

To the right of the empty rail field click on the ... icon. There should be one error cell highlighted. Click **OK** to select it.

Click **OK** at the bottom of the "Load Error Cell" dialog.

The *Error Browser* dialog re-appears. In the top pane select the "Rail" error type. A detailed list of errors appears in the second pane, with the first one automatically selected. In the *LayoutWindow* you should notice that the layout has automatically zoomed to the selected error location.



If you zoom out a little now, you can more easily see the problem: There are standard cells (purple) that are partially placed under the wide VSS ring (METAL3, red) that surrounds the `ram16x128` macros. These macro cells have VSS rails (METAL, blue) along their top edges, but are missing a VDD rail along their bottom edges. These standard cells were inserted after the standard cell rails were created, causing the open VDD connections.

7. Fix the P/G connection problem by routing the standard cell P/G rails:

```
preroute_standard_cells -remove_floating_pieces
verify_pg_nets
```

Note: The P/G connection problems should be cleaned up now.

Task 3. Route and Optimize Design

1. Enable “concurrent” redundant via insertion:

```
# Setting this option prior to routing activates via
# doubling during route_opt, without the need for a
# standalone command. Redundant via insertion can
# optionally also be performed during as an explicit
# step (command) during “chip finishing”
set_route_zrt_common_options    \
   -post_detail_route_redundant_via_insertion medium
set_route_zrt_detail_options     \
   -optimize_wire_via_effort_level medium
```

Note: With `-post_detail_route_redundant_via_insertion` enabled (value set to `low`, `medium`, or `high`), the tool performs redundant via insertion after each detail routing change, including the initial detail routing, ECO routing, and incremental routing. Enabling this option keeps the redundant vias in the design up-to-date with routing changes.
When inserting redundant vias, it is recommended to set the *detail* option `-optimize_wire_via_effort_level` to `medium` (default `low`).

2. Run the following *report* commands to check non-default routing rules and routing setup:

```
report_routing_rules
report_route_opt_strategy
report_route_zrt_common_options
report_route_zrt_global_options
report_route_zrt_track_options
report_route_zrt_detail_options
```

Question 3. In the `global_`, `track_` and `detail_options` reports is *timing driven mode* `true` or `false`?

3. Perform initial routing, which includes *global routing*, *track assignment* and *detail routing*:

```
route_opt -initial_route_only
```

4. Scroll up to the top of the `route_opt` log and locate the "`(ROPT-020)`" information message for *global route*, *track assignment* and *detail route*.

Question 4. Is *timing driven mode* `true` or `false`?

.....

5. Generate post-initial-route reports:

```
view report_clock_tree -summary
view report_clock_timing -type skew
view report_qor
view report_constraints -all
```

Note: There shouldn't be any timing or logical DRC violations. If there were any violations the command in the next step would be executed to optimize for timing and DRCs.

6. Perform post-initial route optimization with `-power` to optimize for power. The `-skip_initial_route` option prevents the initial route from being completely ripped up and re-routed:

```
route_opt -skip_initial_route -power
```

7. Ensure that the logical P/G connections are up to date after routing:

```
derive_pg_connection -power_net VDD -power_pin VDD
   -ground_net VSS -ground_pin VSS
derive_pg_connection -power_net VDD -ground_net VSS \
   -tie
```

Task 4. DRC and LVS Error Checking and Fixing

Check whether there are any physical design rule violations. You will check for these violations using Zroute's `verify_zrt_route` command. The `verify_lvs` command is used to help isolate *opens* and *shorts*.

1. Run the signal route verification tools:

```
verify_zrt_route
verify_lvs
```

Note: There are shorts in the routed design.

2. The next step is to run incremental route_opt to see if that will fix the shorts.

```
route_opt –incremental
```

Note: Incremental route_opt does not help in this case.

3. You can try running ECO route.

```
route_zrt_eco
```

Note: The shorts should be fixed now.

4. Examine the layout for redundant via insertion.
Also, generate the following report and look for this statement near the end: "`Double Via rate for all layers:`"

```
report_design_physical -route
```

Question 5. Do you see double vias on many wires? What conversion percentage has been obtained for all layers?

.....

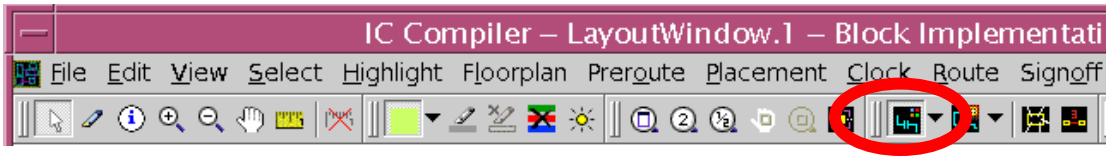
5. Save the design as `route_opt_final`:

```
save_mw_cel -as route_opt_final
```

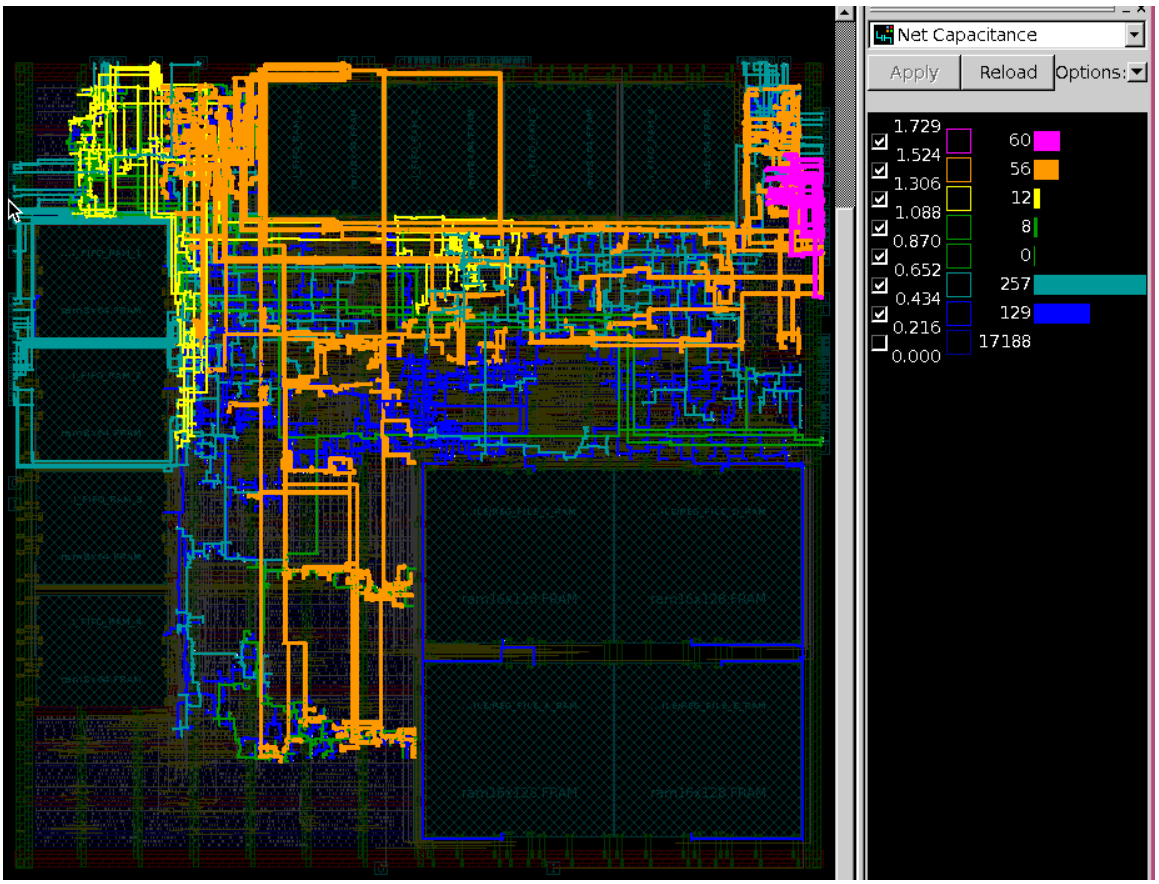
Task 5. GUI Analysis Tools

The GUI provides color coded displays that help you find objects with properties that lie within a certain range of values.

1. Color the display by *net capacitance* values. Click the down arrow next to the *snapshot* button.



Select "**Net Capacitance**". Click "**Reload**". Click **OK** when the "Net Capacitance" dialog pops up. Nets with different capacitance values are highlighted with different color codes.



Note: Click "Apply" to redraw the display after (un)checking any bin.

2. Highlight cells according to their *timing slack.*

At the top of the "Visual Mode" dialog, pull down the menu reading "Net Capacitance" and select "**Cell Slack**". Click "**Reload**", then press **OK** in the new dialog to accept the defaults.

3. In a similar manner, you can highlight cells by *logical hierarchy*. Give it a try to look at the location of various *Verilog* module's cells.

You have successfully completed the Routing lab.

Answers / Solutions

Question 1. Are there any timing or logical DRC violations?

No. There is a maximum area violation, but this is expected since the maximum area constraint (Required Area) is 0!

Question 2. Are there any P/G issues?

Yes, some VDD pins are not connected to the VDD net.

Question 3. In the `global_`, `track_` and `detail_options` reports is *timing driven mode* `true` or `false`?

All three are `false`.

Question 4. Is *timing driven mode* `true` or `false`?

All three are `true`! The `route_opt` command uses its own settings for timing driven mode during routing. It ignores the default setting in the options. If a user explicitly sets the `-timing_driven` mode to `false` in `route_zrt_global_options`, as well as the `track` and `detail` options, the *route_opt* defaults will be overwritten and timing driven will actually be turned off.

Question 5. Do you see double vias on many wires? What conversion percentage has been obtained for all layers?

You should clearly see many double vias. The double via conversion rate averages ~90%.

This page is left blank intentionally.

6 Chip Finishing

Learning Objectives

In the previous labs, the standard cells were placed, the clock trees were synthesized, post-placement timing optimization and routing were completed. In this lab, the design will be taken through the final steps before output to *GDSII* format.

After completing this lab, you should be able to:

- Perform DRC and LVS checks
- Reduce critical areas by performing wire spreading and wire widening
- Fix antenna violations using diodes
- Fill unused locations in the core with filler cells
- Perform optional redundant via insertion
- Perform metal filling for metal density rule compliance
- Stream out *GDSII* data

Lab Duration:

Introduction

The starting cell for this lab has completed the routing phase. The “chip finishing” steps performed in this lab are usually necessary for 90nm technology processes and below. At 65nm and below, these steps are critical in order to maintain high manufacturing yield and chip performance.

Answers / Solutions

There is an *ANSWERS / SOLUTIONS* section at the back of this lab. You are **encouraged** to refer often to this section to verify your answers, or to obtain help with the execution of some steps.

Relevant Files and Directories

All files for this lab are located in the ***lab6_chip_finishing*** directory under your home directory.

lab6_chip_finishing/

orca_lib.mw/CEL

route_opt_final	The ORCA_TOP design after CTS and all routing steps for clocks and signals – the starting point for this lab.

scripts/

cb13_6m_antenna.tcl	Script to set up antenna rules

If you encounter problems, a command script is available to help you recover: ***.solutions/run.tcl.*** You can cut and paste from this file into the command line of
IC Compiler to execute a part of the sequence to catch up.

Instructions

Task 1. Load and Analyze the Design

1. Invoke IC Compiler from the **`lab6_chip_finishing`** directory.

2. Open the design library ***orca_lib.mw***:

```
open_mw_lib orca_lib.mw
```

3. Copy the CEL ***route_op_final*** and rename it as ***chip_finish***:

```
copy_mw_cel –from route_opt_final –to chip_finish
```

4. Open the ***chip_finish*** CEL:

```
open_mw_cel chip_finish
```

5. Verify that there are no DRC violations:

```
verify_zrt_route
```

Note: Notice the Warning "`No antenna rules defined`". These rules will be added and checked later in this lab.

6. Verify that there are no LVS violations (opens, shorts or floating nets):

```
verify_lvs
```

7. Generate a constraint report. There should be no violations:

```
report_constraint –all_violators
# OR
rc
```

Note: If there were any constraint, DRC or LVS violations, they would need to be fixed before continuing with the following chip finishing steps.

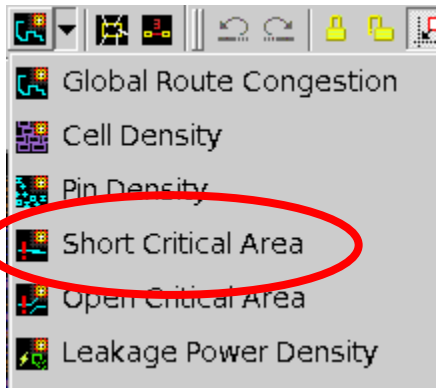
Task 2. Critical Area Reduction

In this task you will analyze the design for potential manufacturing yield loss (critical area) due to opens/shorts, which can be created by airborne particles during the manufacturing process. To reduce the critical area, which can improve yield, you will optimize the routing by spreading and/or widening wires.

1. Open the GUI windows (if not already opened):

```
start_gui
```

2. Analyze the critical areas graphically:

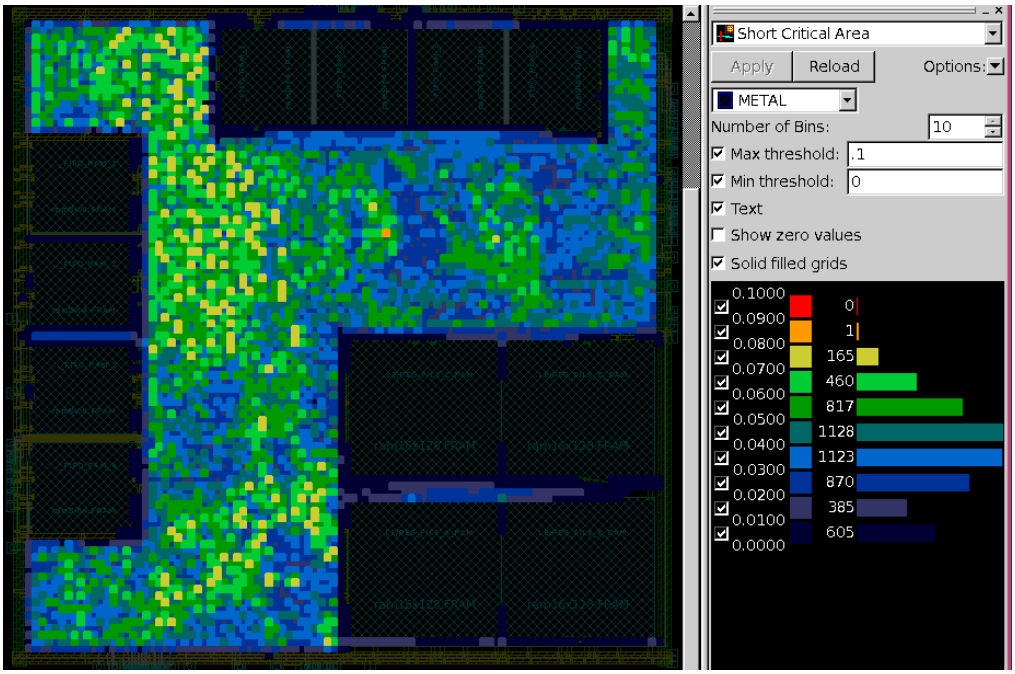


Select "**Short Critical Area**"
Select **Reload** then **OK**.

This executes `report_critical_area -fault_type short`.

Set the "**Max threshold**" value to **0.1**, then click **Apply**.

You should see a heat map distribution similar to the one below. From this map we see that the maximum critical area "ratio" (the critical area divided by the area of the small square "analysis window") is below 10%, which is relatively low.



3. There is also a textual report that is automatically created in the current working directory, called `output_heatmap`. Since this file is over-written when you perform another critical area analysis, you should save a copy:

```
sh mv output_heatmap cca.short.before.rpt
```

4. Close the *Critical Area* heat map.

5. Perform wire spreading to reduce the critical area for *shorts*:

```
spread_zrt_wires
```

6. Generate another *short heat map* – remember to **Reload** to execute a new analysis. The critical area improvement is not noticeable from the heat map since this design didn't have much of the potential problem to start with. Close the heat map.

7. You can, however see the effects of wire-spreading by zooming in and taking a closer look. It is easier if you focus on one layer at a time, for example METAL3:

   In the *View Settings* panel (press [F8] if the panel is not visible) select the *Objects* tab, then enable **Track** visibility and turn off **Cell** visibility. Select the *Layers* tab and select **Hide All** to turn off visibility of all layers. On the METAL3 row click on the **Shape** and the **Track** columns. Zoom in until you see the dotted track lines. You should now be able to locate wire segments that have been pushed off their routing tracks wherever possible, in order to increase spacing between the metal traces.

8. Again, you should save a copy of the critical area report. You can analyze these files after you complete the lab, if you like:

```
sh mv output_heatmap cca.short.after.rpt
```

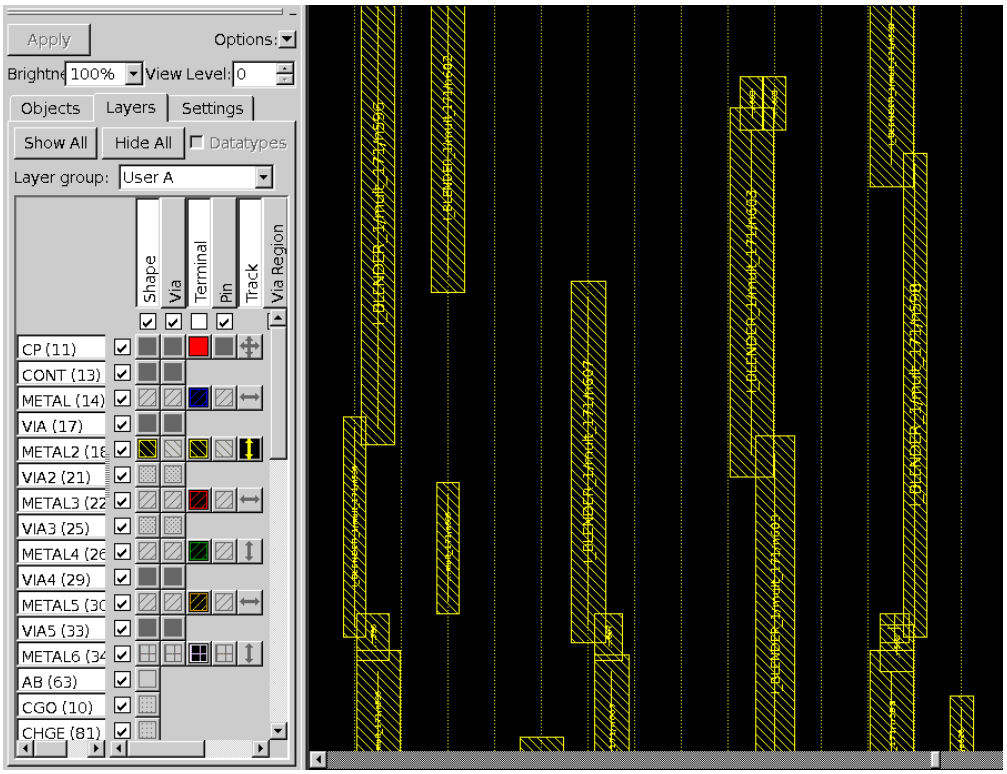
9. Now perform wire widening to reduce open critical area. Also capture the "before and after" critical area outputs:

```
report_critical_area -fault_type open
sh mv output_heatmap cca.open.before.rpt

widen_zrt_wires

report_critical_area -fault_type open
sh mv output_heatmap cca.open.after.rpt
```

10. Close the Critical Area heat map if it is still open.

Here is a screen shot showing the results of wire spreading and wire widening

on METAL2. You can see that some segments are wider than the default and some are pushed off-track.



11. Verify that the design has no DRC, LVS or constraint violations, then save the cell as `chip_finish_ca`:

```
verify_zrt_route
verify_lvs
rc
save_mw_cel –as chip_finish_ca
```

Note: If there were any timing or logical DRC violations at this point you could execute `psynopt` with the appropriate optimization focus (`-only_design_rule`, `-in_place_size_only`, `-size_only`, etcetera). If there were any physical DRC violation you would execute an incremental detail route:
`route_zrt_detail -incremental true`

Task 3. Fixing Antenna Rule Violations with Diodes

1. To install a set of antenna rules, source the following command file:

```
source –echo scripts/cb13_6m_antenna.tcl
```

The script sets up antenna layer rules for all conductors and via layers.

2. Use your editor to examine the setup script you just sourced.

The *"set_parameter -module droute -name doAntennaConx -value 4"* command sets antenna reporting/fixing to an 'advanced' mode. This instructs the router to use advanced algorithms as well as the antenna rules defined later in the script by *define_antenna_rule* and *define_antenna_layer_rule*.

These antenna rule commands set up modes which determine how metal layers are processed, as well as default and specific values for antenna ratios. Man pages for the *rule* commands are included in the script file. Take a look if you want additional information.

3. Check the current antenna ratio rules that have been defined, and check if any antenna violations exist in your design:

```
report_antenna_rules
verify_zrt_route
```

Question 1. Are there any antenna violations?

.....

4. Enable antenna fixing with diodes, then perform incremental detail route to implement the diode insertion:

```
set_route_zrt_detail_options \
   -insert_diodes_during_routing true
route_zrt_detail -incremental true
```

5. For instructional purposes, check for LVS violations at this time:

```
verify_lvs
```

Question 2. Can you explain the cause of the VSS and VDD shorts?

.....

6. The inserted diodes need to be logically connected to power and ground. We could wait until the end of the chip-finishing flow, where we will perform a final "pg connection", but we will do it here as well, to confirm that this is indeed the cause of the LVS errors:

```
derive_pg_connection -power_net VDD -power_pin VDD \
                     -ground_net VSS -ground_pin VSS
```

```
derive_pg_connection -power_net VDD \
                     -ground_net VSS -tie
```

7. Verify that the design has no timing, DRC (including antenna), or LVS violations, then save the cell as `chip_finish_antenna`:

```
verify_zrt_route
verify_lvs
rc
#If there were any timing or logical DRC violations
use below commands
route_opt –incremental
derive_pg_connection -power_net VDD -power_pin VDD \
                     -ground_net VSS -ground_pin VSS
derive_pg_connection -power_net VDD \
                     -ground_net VSS -tie
verify_zrt_route
verify_lvs
rc

save_mw_cel –as chip_finish_antenna
```

Task 4. Insert Standard Cell Fillers

Filler cells are used in the core area to ensure continuous power and ground rail connections, as well as continuous *n*- and *p-well*s in each row of standard cells. Some libraries may contain filler cells with a P/G bypass capacitor onboard. These usually contain metal shapes. Insert these filler cells after all routing is complete, but before the metal fill step.

1. Note that the library contains three different *filler cells*. Two contain metal and one does not. You want to insert cells with metal first, then follow with non-metal cells:

```
insert_stdcell_filler  \
    -cell_with_metal "feedth9 feedth3"  \
    -connect_to_power VDD -connect_to_ground VSS \
    -between_std_cells_only
```

Note: You can ignore a warning message about the feed cells not a std filler cell subtype. To avoid the warning messages, the library need to be updated.

2. Use the *View Settings* panel to turn off all metal layer visibility and ensure that **Cell** visibility in on. Examine the layout to locate some filler cells.

Question 3. What are the names of the filler cell instances?

.....

Question 4. What happens to an inserted filler cell with metal if it causes a DRC violation? Hint: look for messages in the log.

.....

3. Now insert non-metal filler cells:

```
insert_stdcell_filler -cell_without_metal "feedth" \
    -connect_to_power VDD -connect_to_ground VSS \
    -between_std_cells_only
```

4. Make sure your design is still “clean”:

```
verify_zrt_route
verify_lvs
```

```
rc
```

Task 5. Insert Redundant Vias

You may have noticed that most of the vias are already doubled as a result of *concurrent via doubling* during detailed route. This task of redundant via insertion is optional during chip finishing.

1. Generate a physical design report to determine the existing double via rate:

```
report_design_physical –route
```

Question 5. What is the current double via rate for all layers?
Hint: Look near the end of the report.

.....

2. Generate a via mapping table that will be used automatically during redundant via insertion:

```
insert_zrt_redundant_vias  -list_only
```

If you need to change via mapping from the default, you need to use `define_zrt_redundant_vias` command as shown in the lecture. You will use the default values for this lab.

3. Use the default *medium* effort to insert redundant vias:

```
insert_zrt_redundant_vias  -effort medium
```

4. If you generate a new physical design report you will notice that there is very little improvement in the percentage of redundant vias. This is because most of the vias that could be doubled were already done during routing. Running with *high* effort would yield a higher via doubling rate but it will also cause many jogs to fit the additional redundant vias. Too many jogs (corners) may not be good for lithography in advanced process technologies.

5. Make sure your design is “clean”:

```
verify_zrt_route
verify_lvs
rc
```

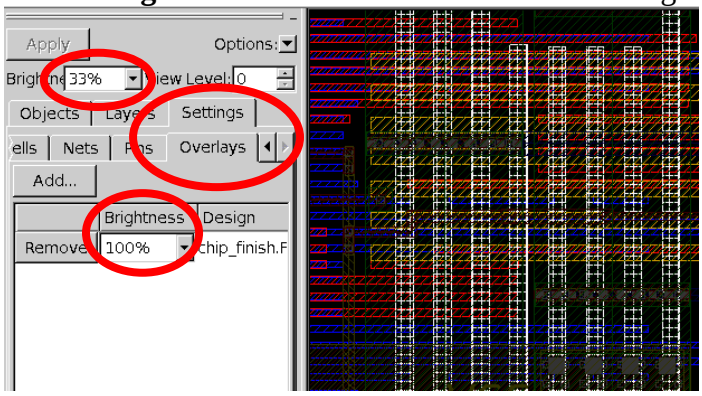
6. One of the very last steps before final physical verification is *metal filling*, to meet *metal density rules*. This will use up most of the remaining routing resources, so it must be executed after all routing and related optimizations and antenna rule fixing.

For actual designs it is recommended that you use either *Hercules* or *IC Validator* to perform sign-off metal fill. In this class you will use IC Compiler's *insert_metal_filler* to fill unused tracks with metal shapes:

```
insert_metal_filler  -routing_space 2 –timing_driven
```

7. The metal filling goes into a *FILL view* and is not visible until you "overlay" this view on top of the layout view. To overlay the metal fill:

Select the **Settings** tab from the *View Settings* panel.
Select **Overlays** tab (you may need to click on the right-arrow to see the tab).
Click the **Add** button.
Select **chip_finish.FILL** and click **OK**.
Set the **Brightness** of the FILL view so that it is brighter than the CEL view.



8. Ensure that all P/G pins of any cells that were added during this chip-finishing phase are logically connected to the P/G nets:

```
derive_pg_connection -power_net VDD -power_pin VDD \
                     -ground_net VSS -ground_pin VSS
derive_pg_connection -power_net VDD \
                     -ground_net VSS -tie
```

9. Perform final DRC, LVS and timing analysis:

```
verify_zrt_route
verify_lvs
rc
```

10. With chip finishing completed, the physical design is done. You can now write out the design in *GDSII* or *Oasis* format for "tape-out" using the `write_stream` command (which can be configured, if needed, using `set_write_stream_options`):

```
save_mw_cel -as chip_finish_final
close_mw_cel
write_stream -cells chip_finish_final orca.gdsii
close_mw_lib
```

11. Exit IC Compiler.

You have completed the last lab of the IC Compiler 1 Workshop.

Answers / Solutions

Question 1. Are there any antenna violations?

Yes. Over 200 violations are reported.

Question 2. Can you explain the cause of the VSS and VDD shorts?

The diode cells that were placed in the standard cell placement rows have VSS and VDD pins. These P/G pins are touching the P/G rails along the top and bottom of the placement rows. Since we have not yet established a "logical" connection between the new P/G pins and the VSS/VDD nets, LVS sees these touching metals as shorts.

Question 3. What are the names of the filler cell instances?

`xofiller!feedth9!###` and `xofiller!feedth3!###`.

Question 4. What happens to an inserted filler cell if it causes a DRC violation?

It is deleted.

Question 5. What is the current double via rate for all layers?

It is around 90%.

This page is left blank intentionally.