



CUSTOMER EDUCATION SERVICES

IC Compiler II: Block-level Implementation Workshop

Lab Guide

20-I-078-SLG-009

2018.06-SP4

Synopsys Customer Education Services
690 E. Middlefield Road
Mountain View, California 94043

Workshop Registration: <https://training.synopsys.com>

Copyright Notice and Proprietary Information

© 2019 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>. All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Document Order Number: 20-I-078-SLG-009
IC Compiler II: Block-level Implementation Lab Guide

0

IC Compiler II GUI

Learning Objectives

This lab's purpose is to familiarize you with the IC Compiler II GUI.

After completing this lab, you should be able to:

- Invoke and exit the IC Compiler II GUI
- Navigate the *layout view*
- Control object and layer visibility
- Select and query layout objects
- Control the view level
- Rearrange panels in the GUI
- Use the *Recent* menu and *Favorites*
- Use *Command Search*, as well as *help* and *man* to get help and additional information about commands and options



Lab Duration:
45 minutes

Lab 0

Task 1. Launch IC Compiler II

1. Log in to the *Linux* environment with the assigned *user id* and *password*.
2. From the lab's installation directory, change to the following working directory and invoke IC Compiler II:

```
$ cd lab0_gui  
$ icc2_shell
```

The *Linux terminal* prompt becomes `icc2_shell>`, the *IC Compiler II* shell command prompt.

3. Some of the Linux commands also exist at the IC Compiler II prompt. Have a look at your current directory:

```
icc2_shell> ls
```

You will see that command and output log files were created (`icc2_shell.cmd.*` and `.log.*`, with date/time). The `.cmd` file records all commands, including initialization commands invoked during start-up. The `.log` file records commands and command output after tool start-up. In addition, there is an `icc2_output.txt` file that also contains all output. Do not spend too much time looking at the log file contents.

Note: Log/cmd file naming is defined through variables in the initialization file, `.synopsys_icc2.setup`.

4. Start the GUI:

```
icc2_shell> start_gui
```

The *IC Compiler II BlockWindow* opens.

5. Click on the symbol to open an existing block:

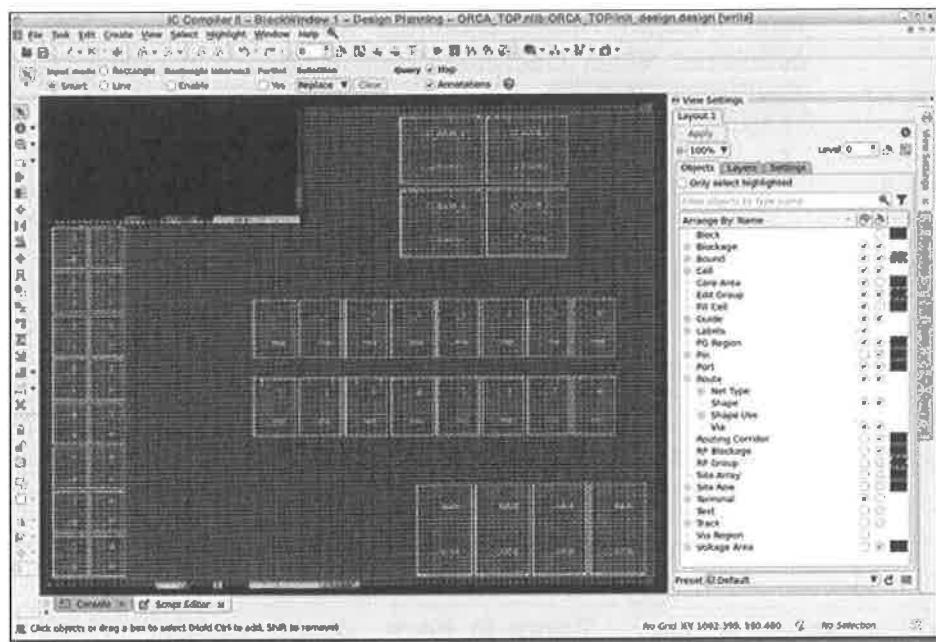


Open an existing block

6. In the dialog that opens, click on the yellow symbol in the top-right corner to **Choose** a design library called “`ORCA_TOP.dlib`”.

Design libraries are marked with this symbol:

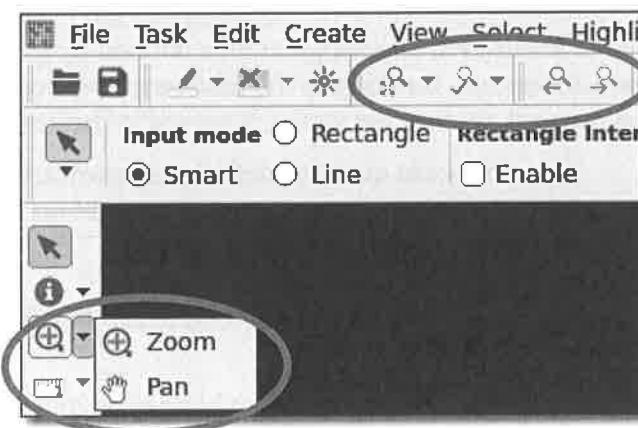
7. Now that a library has been chosen, select “**ORCA_TOP/placed**” from the list at the bottom of the **Open Block** dialog. Click **OK**. You will be presented with the layout view of the design.
8. Enlarge or maximize the GUI window. You will see that the layout view adjusts to fit the larger window.



You are looking at the layout of a placed design: Macros are placed at pre-defined locations, the power mesh (vertical and horizontal VDD/VSS *straps*) has been completed, and standard cells have been placed.

Task 2. Navigating the Layout View

1. Spend a few minutes to become familiar with the *zoom* and *pan* buttons.
- Hint:** A short, descriptive *ToolTip* will pop up when a mouse pointer is held motionless over a button.



Lab 0

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

2. Use *hot keys*: Lower-case [F] or [Ctrl F] both correspond to *zoom fit all* (or full view), for example. [+] or [=] is zoom-in 2x, [-] is zoom-out 2x.
3. Find out about other hot key definitions by selecting the pull down menu **Help → Report Hotkey Bindings**. A new view appears, listing the hot key definitions. When done, close the *Hotkeys* view by clicking on the tab's "X".

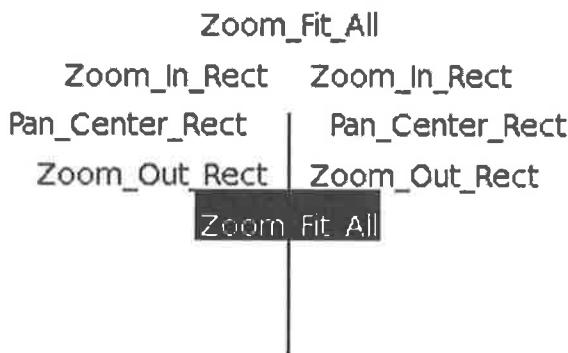


Note: *Hot keys* can be defined by using `gui_set_hotkey`. In the interest of time, do not attempt this presently.

4. You can also 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: Lower-case [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.

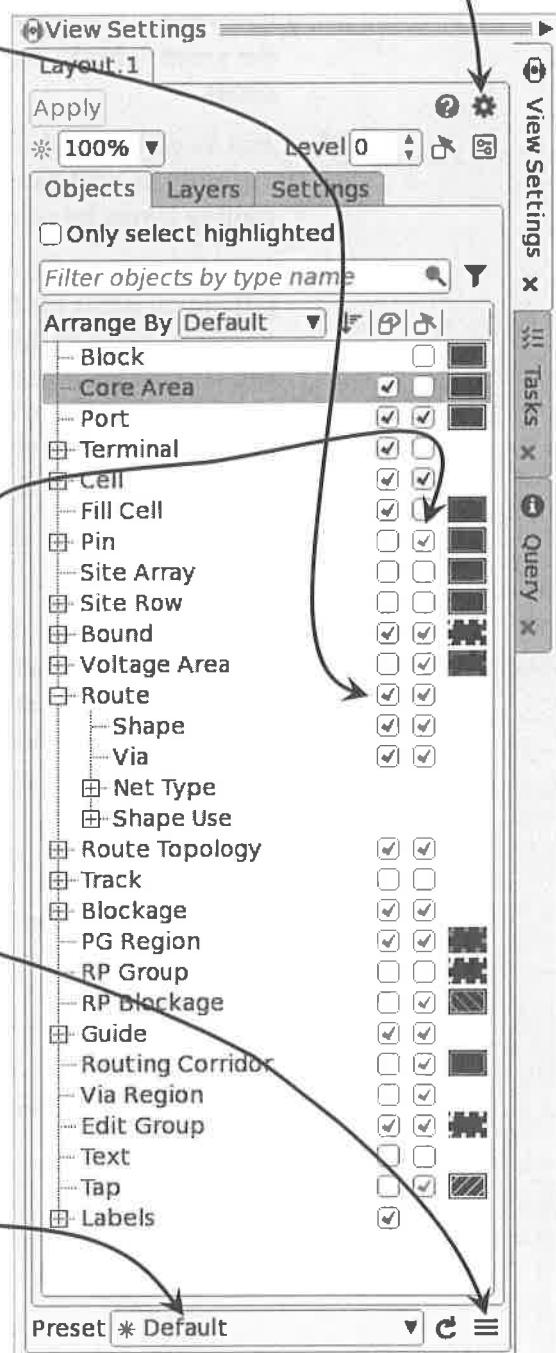
Note: You can query or define your own strokes by using the commands `get_gui_stroke_bindings` and `set_gui_stroke_binding`.

5. The keyboard **arrow keys** can also be used to pan the display up/down/left/right. 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 Object and Layer Visibility

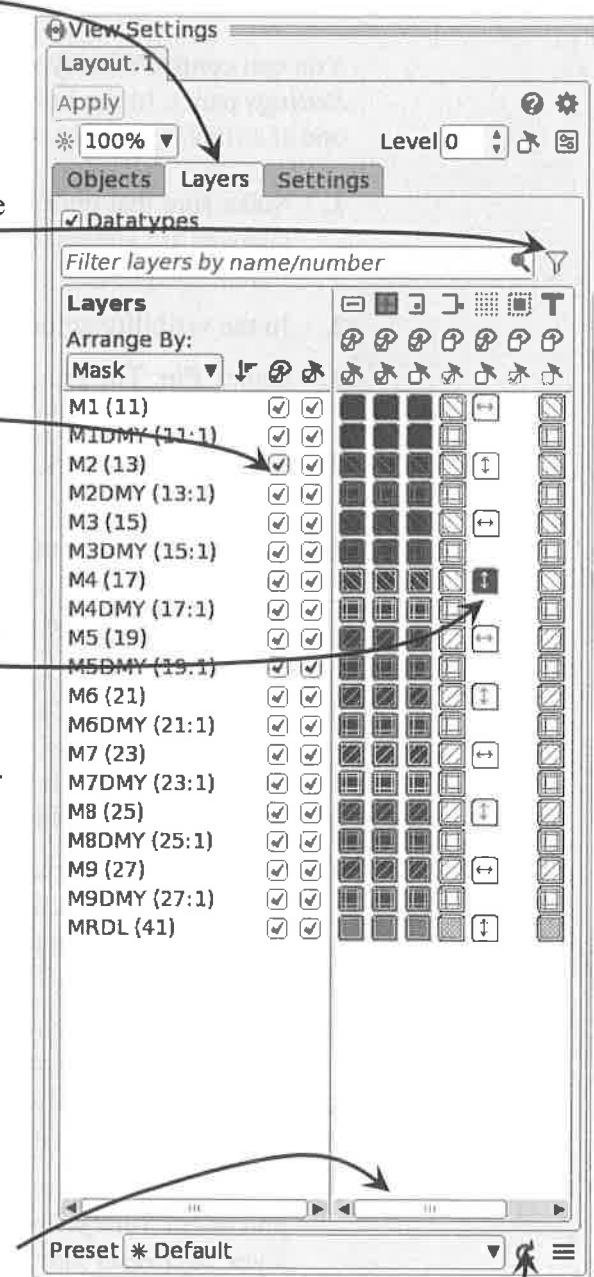
You can control what types of objects are *visible* and/or *selectable* through the *View Settings* panel. In the following steps you will turn on visibility to some key objects one at a time, to clearly see what they represent.

1. Make sure that under the *Options* button, *Auto apply* is checked: This way changes are applied immediately without having to confirm with the *Apply* button each time.
2. In the visibility column uncheck **Route**.
3. Check **Pin**. The input, output and power pins of the cells are displayed.
4. Uncheck then check **Labels**. This controls cell name visibility.
5. Zoom in to one of the macro pins. Expand **Labels** by clicking on the + icon on the left. Check **Pin**. Pin names are now visible as well.
6. In selection mode (press [ESC]), draw a box to fully enclose a few macros. This selects all selectable objects inside the box, which are highlighted in white.
7. Make pins un-selectable by unchecking the box in the selection column. Draw the same box again to see that only the macros are selected.
8. Check **Route**. All routes are displayed. Expand **Route** and you will find that you can control visibility of routes by *Net Type*. (e.g. *Clock*, *Ground*, etc.).
9. Save the view settings. Click this button and select *Save preset as...*
Type *MyPreset* into the *preset name* field and click **OK**. *ICC II* creates a file *MyPreset.tcl* in the *~/synopsys_icc2_gui/presets/Layout* directory. All saved presets will be loaded automatically whenever *ICC II* is started again.
10. Select **Default** from the pull down list to restore the default settings. Fit the view to the window [F].



Lab 0

11. Select the **Layers** tab, which can be used to fine-tune the routing visibility further on a layer-by-layer basis.
12. To list only the actual routing layers in the *Layers* tab, click on the filter button and uncheck “All Layers” first to clear everything, then check “Routing”. Click the green “check” mark to apply the filters.
13. Just as with the *Objects*, you can control visibility as well as selectability of the routing layers by using the checkmarks.
14. In addition, you can click on the individual items for finer control. For example, click on the up/down arrow next to M4 to toggle track visibility. In addition, you can control the visibility of Shape, Via, Terminal, etc. (*terminals* are the physical pins of the top-level block ports)
15. Zoom into the layout and turn the visibility of various layers off and on, to get a better understanding of the PG mesh (made up of wide horizontal M7, and vertical M8 straps, as well as narrow vertical M2 straps).
16. Notice that there are more columns which you can control. Use the horizontal scroll bar to see them all.
17. Re-apply the *Default* settings by clicking the “*reload*” circular arrow.



Task 4. Querying and Selecting Objects

1. To be able to query or 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 (Select Tool)** button  or press the [Esc] key.
2. Back in the **Objects** tab, uncheck **Route** Visibility.
3. Hover the pointer over an object without clicking the mouse. The object is highlighted with dashed white lines, and an *InfoTip* box appears in the bottom-left, displaying some key attributes of the object.
4. Now select a single object with a left mouse click. The selected object is highlighted with solid white lines, and remains highlighted until de-selected, or a different object is selected. Keep the object selected.
5. While one object is selected, hover the mouse cursor over a different object. Notice that the *InfoTip* box displays information about the *dashed white line* object, not the selected (solid white line) object.
6. To obtain a full query of a selected object press lower-case [Q] or use the menu: **Select → Query Selection**. A query panel lists all the attribute values of the selected object.
7. You can return to the *Objects* tab of the *View Settings* panel by selecting the **View Settings** tab.
8. Deselect all objects by either clicking on an empty area in the layout, by using the menu **Select → Clear**, or by typing [Ctrl D].
9. Select multiple objects in the same area with a left button *drag-and-draw*. All selectable objects within the drawn rectangle are selected.
10. Keep what is selected and select additional objects by holding down the [Ctrl] key while selecting with the left mouse click.
11. Enable **Route** visibility, zoom into a small area, then click on an area with multiple objects stacked on top of each other (for example, a via connecting a horizontal and a vertical metal route). Notice that one object will be selected (solid white), while a different object will be queried (dashed white). The *InfoTip* box goes with the *dashed* object.
12. Cycle through the *stacked* objects by repeatedly clicking the left mouse button, without moving the cursor: Notice that both the solid and dashed line objects cycle. Alternatively, press F1 to cycle through just the object queries.
13. If it is difficult to notice the highlighted 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. 



Lab 0

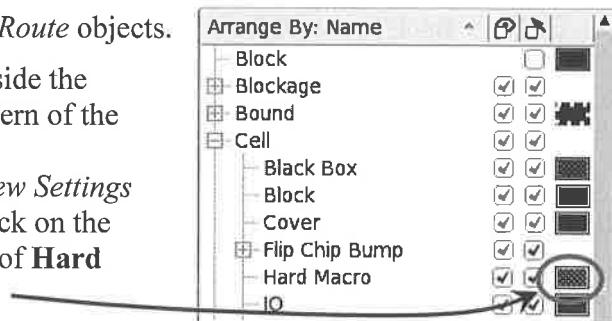
Task 5. Controlling the View Level

By default, the GUI displays all shapes at the current block level, e.g. metal shapes, blockages, pins etc. To see shapes inside standard cells or hard macros, you have to increase the view level, and enable them to be “expanded”.

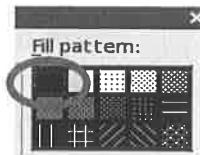
1. Disable the visibility of all *Route* objects.

2. To better see the objects inside the macros, remove the fill pattern of the macro shapes:

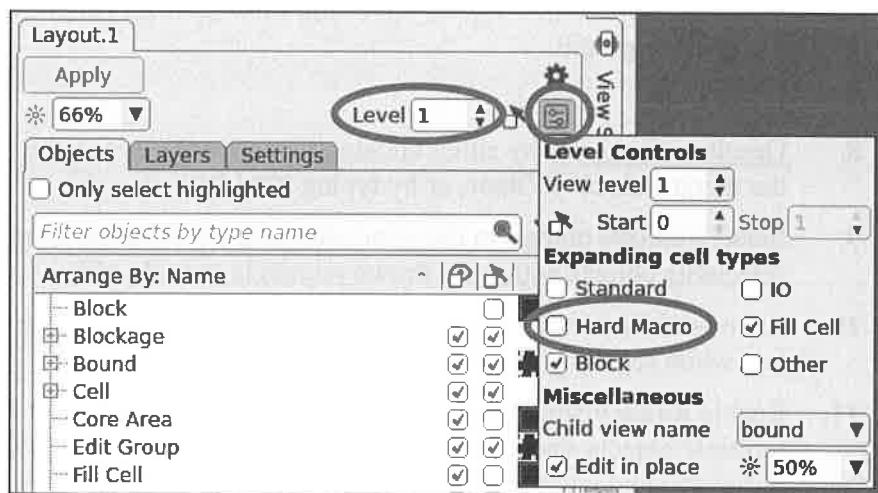
In the *Objects* tab of the *View Settings* panel, expand *Cell*, then click on the aquamarine outline pattern of **Hard Macro**.



In the *Select Style* dialog, go to the *Fill pattern* section in the upper-right, and select the **black** (no fill) pattern in the upper-left, then click **OK**.



3. Increase the viewing *Level* from 0 to 1, then click on the *Hierarchy Settings* button to the right of the *Level* field, and check **Hard Macro**.



If you zoom into one of the RAMs you should see the structures inside the macro. You can turn layer visibilities off/on to see individual layers. Since these are frame views, what you are seeing are mostly routing blockages.

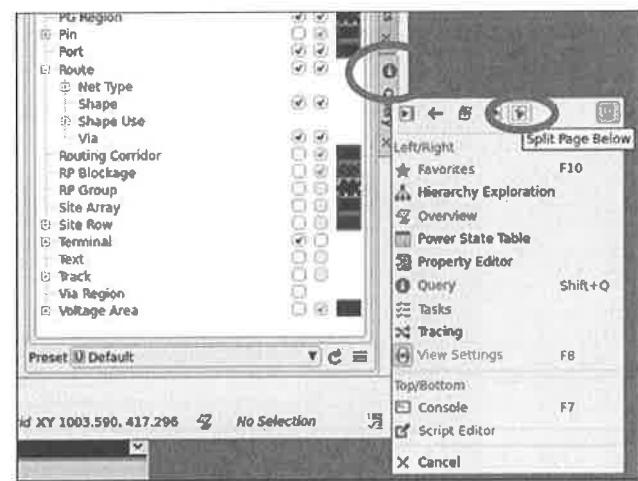
4. You can also interact with structures within cells: Click on the *Multiple Levels Active* button shown here. Now you can hover over objects within cells, and analyze them. This functionality is used extensively in hierarchical design planning, where you can perform physical manipulation at multiple physical design levels at the same time.



Task 6. Rearranging Panels

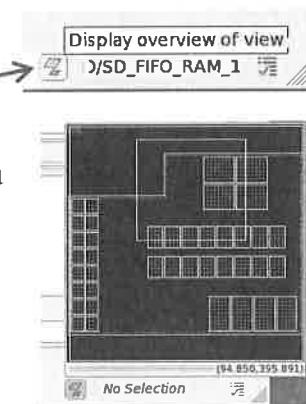
As you may have noticed, you can switch back and forth between the panels (or tabs) on the right. Sometimes it can be useful to display more than one panel at the same time. This is particularly useful if you have a lot of screen real estate.

1. The query panel should still be open. If not, select a macro then press **q** to bring it up again.
2. Right-click on the query tab – a context menu should be displayed.
3. From that context menu click on the “**Split ‘Query’ Below**” button as shown on the right. The panel area on the right will now split into a top and a bottom area, with the query panel being in the bottom area.
4. Now that you have two panel areas, you can drag and drop tabs from one area to the other. For example, try dragging the *Favorites* tab (explained in the next Task) from the top to the bottom area. Of course, you can also rearrange the tabs within the same panel area using drag and drop.
5. When you right-click on the empty area under the tabs, or on a tab (as you’ve done above), you can also open new panels, for example the *Property Editor*, which can be used to display or change attributes of selected objects.
6. Try the other menu banner icons, like **Detach**, **Dock** and **Collapse**. Also note that you are not limited to just two panel areas – you can split the panels again.



7. In the bottom-right of the *BlockWindow*, click on the **Display overview of view** icon, as shown here:

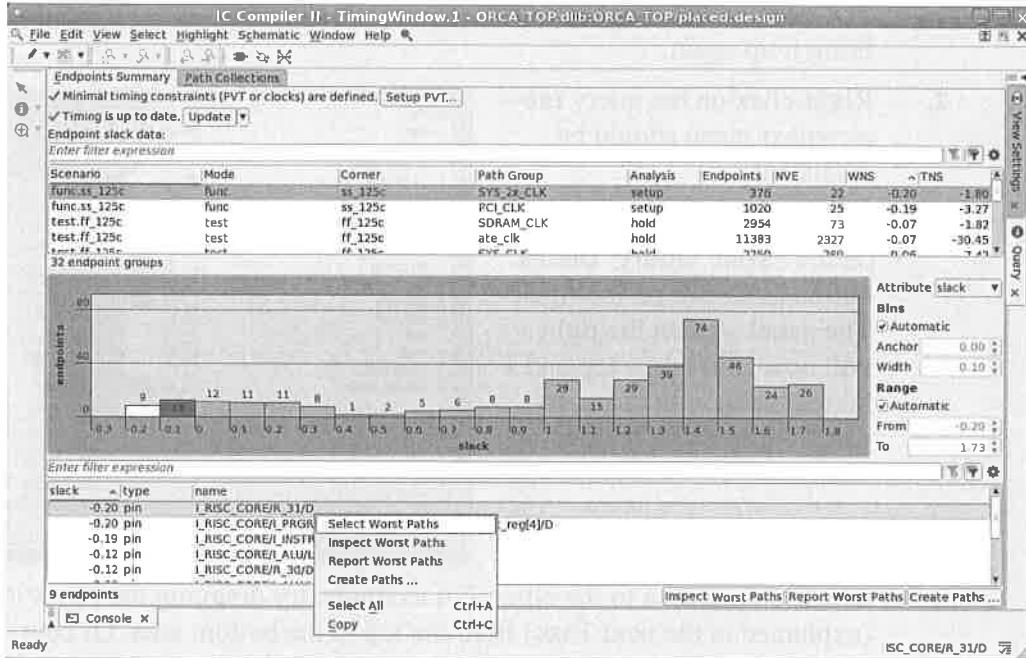
The *Overview* panel is a miniature version of the layout view. Zoom in to an area in the layout, and you will see a yellow box outlining that area in the overview panel. You can move and resize that box within the overview panel and the layout view will adjust accordingly. Note that the overview window is on-demand – it closes once you click anything in the full layout window.



Lab 0

Task 7. Timing Analysis

Have a look at the timing of this design. Use the menu **Window→Timing Analysis Window**. In the new window that appears, press the **Update** button in the top-left to update the timing.



Once timing is up-to date, you will see a histogram for the timing distribution. Violating paths show up as red bars on the far left, and positive slack paths are green.

Click on the left-most red bar and you should see timing end-points displayed on the bottom. Select the first entry, then right click. From the context menu, choose “Select Worst Paths”. You will see the timing path in the layout view.

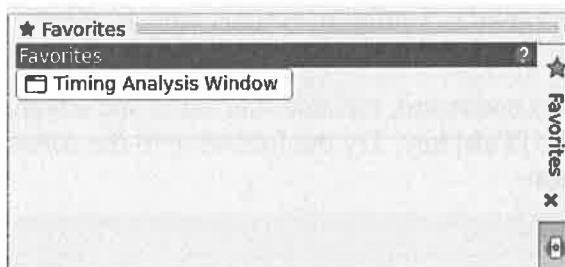
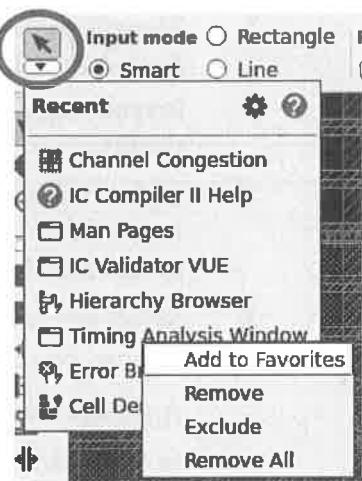
Task 8. Using the Recent Menu and Favorites

Whenever you perform a GUI function, like viewing a congestion map or analyzing timing, this function can be repeated by using the **Recent** pull-down menu, as shown here (top-left in the GUI, under *Select Tool*).

In addition, if you have certain functions that you use over and over, it might make sense to add them to your *Favorites*.

Pull down the *Recent* menu, then right click on a function, and select “Add to Favorites”.

You will see this function listed in the *Favorites* pane as shown here:



Lab 0

Task 9. Getting Help

1. The quickest way to get help is to use *Command Search*. Access *Command Search* by using the keyboard shortcut lower-case [H] or by clicking on the magnifying glass  at the top of the block window, on the right end of the menus.
2. In the “Search for commands” field, begin typing “place”. Notice that as you type, ICC II lists all the menus, commands and application options that relate to that search string.
3. Select one of the **Application Options**, for example, `place.coarse.congestion_analysis_effort`.
A second window will open, titled “Application Options”.
In this window you can change the settings of application options, or you can search for them. Explore this window a little to become familiar with it.
Application options are used to configure various aspects of how *IC Compiler II* works. You will learn more about them in the next lecture.
4. *IC Compiler II* supports command, variable, file name and command option completion through the **[Tab]** key. Try the following in the *console* at the `icc2_shell>` prompt:

```
h[Tab]e[Tab] he[Tab] -v[Tab] [Enter]
```

5. To view the *man* page on a command or application option you need to enter the exact command or variable name. Alternatively, you can enter the starting characters of a command and use *command completion* to find the rest (auto-completion is not available for application options). If you are not sure what the exact name is, use `help` for commands, use `get_app_options` or `report_app_options` for application options, and use `printvar` for variables, along with the * wildcard. Here are some examples:

Let's say you are looking for more information about the clock tree synthesis command, but you do not remember the exact command name. 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 in, namely, `synthesize_clock_trees`.

Of course, you could have entered `syn` in *Command Search* as well.

6. To list the available options for `synthesize_clock_trees` (note that you can also use tab completion on the `synth*` command after `help`):

```
help synthesize_clock_trees -verbose  
or  
help synthesize_clock_trees -v  
or  
synthesize_clock_trees -help
```

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

```
man synt [Tab]c [Tab]e [Tab]  
or  
man synthesize_clock_trees
```

8. Now let's say you need help on a specific application option, but again, you don't remember its exact name, but it pertains to *CTS*. To list all application options that start with "cts", enter:

```
report_app_options cts*
```

From the list you can identify the option of interest.

Notice that the `report_app_options` command also lists the current value of each option. This report output will be explained in an upcoming lecture.

9. To get a full help manual page of the application option, type:

```
man cts.compile.enable_cell_relocation
```

10. You can also get additional help for an error or warning message, using the unique message code, for example:

```
man ZRT-536
```

Lab 0

11. Finally, specifically for this workshop, we are providing a few custom functions for the shell that can simplify your work. They are defined in the file `../ref/tools/procs.tcl`. Try the following:

```
aa syn
```

The `aa` function searches through commands, application options and variables that match the given string. In addition, it shows the current value.

If you want to redirect content to a separate window, try these:

```
v man ZRT-536  
v aa cts
```

`v` is a user-defined alias that calls the “`view`” custom function, useful for viewing long reports, and allows for regular expression searches.

12. To list the workshop-provided helper functions and aliases, type “`ces_help`”.
13. Quit *IC Compiler II* by using the menu **File→Exit**, or by typing `exit` at the command prompt, and selecting **Discard All**.

You have completed the IC Compiler II GUI lab.



2

Floorplanning

Learning Objectives

The purpose of this lab is to familiarize you with the basic block-level floorplanning operations in IC Compiler II.

After completing this lab, you should be able to:

- Define a rectilinear block shape
- Place voltage areas inside the block
- Perform placement to determine macro locations
- Define block pin locations
- Analyze congestion and netlist connectivity
- Run PG prototyping
- Source a script that builds a power network using Pattern-based Power Network Synthesis



Lab Duration:
35 minutes

Lab 2

Introduction

ORCA Design

The example design used in all labs is called ORCA which was specifically created to address the needs of training. The design was created by the Synopsys *Customer Education Services* department.

SAED32nm Library

The technology library used in this workshop is a 32 nm library which can be freely distributed amongst *Synopsys* customers. It was created by *Synopsys Inc.* to serve as a means of demonstrating the various needs of modern high-frequency, multi-voltage, multi-scenario designs. For more information about this library please visit <http://www.synopsys.com>.

This Lab

You will be creating a basic floorplan for the ORCA_TOP design. This will include creating the initial outline of the block, shaping and placing the voltage areas, placing the macros, performing analysis on macro placement, and finally implementing the power network.

Instructions

Answers / Solutions

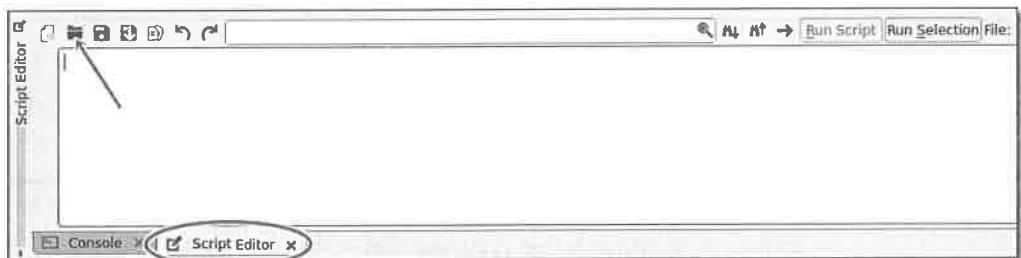
You are encouraged to refer to the end of this lab for answers and help.

Task 1. Invoke *ICC II* and load the ORCA_TOP Block

1. Change your current directory to `lab2_floorplan`, then invoke *IC Compiler II*:

```
UNIX% cd lab2_floorplan
UNIX% icc2_shell -gui
```

2. Select the **Script Editor** tab at the bottom-left of your *ICC II* window. See the circled icon below.



3. Click on the “folder” button (see the arrow above) and *open* the `run.tcl` file.

This file contains the commands that you will be executing in this lab. Instead of opening the file in a separate editor and using copy/paste to transfer commands, you can use the built-in script editor to simply highlight/select lines you want to execute, and then click on the **Run Selection** button.

Try this now by selecting the entire line `echo "hello world"`, then clicking **Run Selection**. Look at the results in the shell window.

4. Open the block which has been initialized with Verilog, UPF and timing constraints, ready to be floorplanned. Use the GUI (**Open an existing block**), or select/run the following command:

```
open_block ORCA_TOP.dlib:ORCA_TOP/floorplan
```

Lab 2

Task 2. Initial Floorplanning using the Task Assistant

For this task you will use the *Task Assistant*. You do not need to run commands from `run.tcl` for the next few tasks.

1. Open the *Tasks* palette (if not already open) by right-clicking your mouse anywhere in the tab area on the right (where the *View Settings* palette is), then selecting **Tasks**. In the pull-down menu at the top of the *Tasks* palette, select **Design Planning**.

Note: You could also use the full *task assistant* by pressing **F4**, or using the menu **Task→Task Assistant**. Then, select the “>” (*Show task navigation tree*) icon in the upper-left corner of the *Task Assistant* window, and in the pull-down

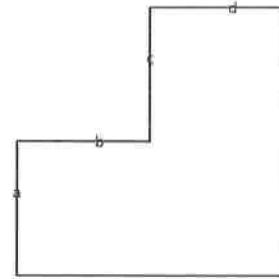


menu choose **Design Planning**

2. Select **Floorplan Preparation →Floorplan Initialization**.

This opens the *Floorplan Initialization* dialog.

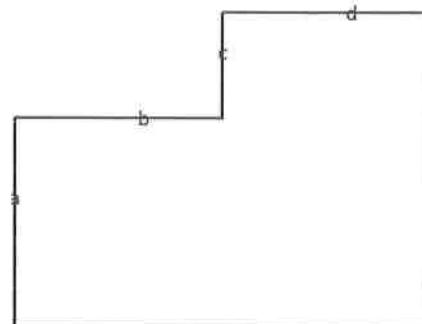
3. Use the boundary **Type** and **Orientation** pull-down fields to create the block shape shown here (click on **Preview** under the black *Display Window* in the dialog to double check your set up):



4. Verify that the **Side size control** is set to **Ratio**.

5. Enter the correct **Sides** numbers so that side “c” is half the size of sides “a”, “b” and “d” (which are equal).

Hint: Use whole numbers only. See the bottom of this page if you get stuck.

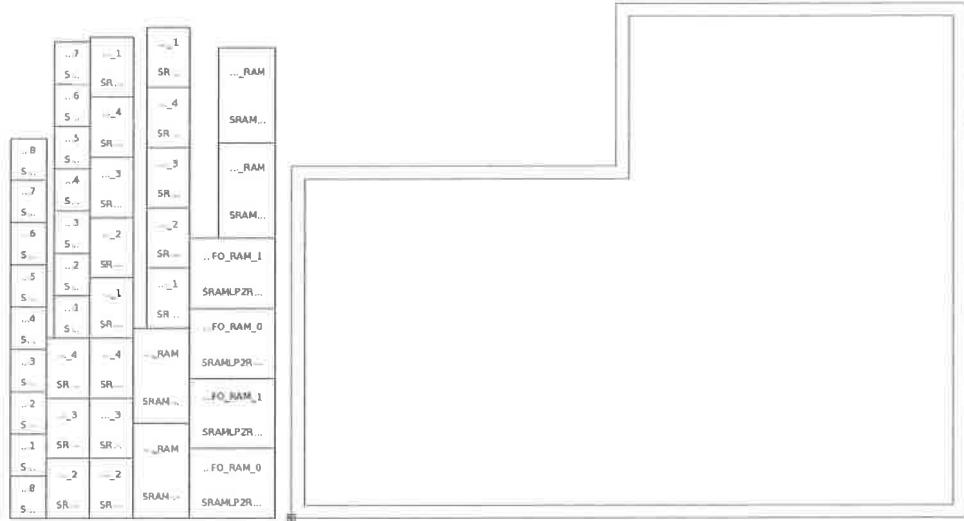


6. **Preview** the floorplan, and once it looks correct, specify a **Uniform** spacing value of **20** between the core and the die.

7. Click on **Apply** to generate the initial floorplan.

Core shape should be set to “*L-shape*”, *Orientation* should be set to “*W*”.
Set the 4 parameters as follows: $a=2$, $b=2$, $c=1$, $d=2$

8. You should now see something like this:



Task 3. Block Shaping

1. From the **Tasks** palette, select **Block Shaping** → **Block Shaping**: This is used to automatically create (shape) *voltage areas*.
2. Select the second **Shape Blocks** tab to the right of the **Shaping Options** tab, and select **Apply**.
3. After a short moment, the operation should complete. Bring the layout view to the foreground.
4. Enable visibility of voltage areas by selecting **Voltage Area** from the **View Settings** → **Objects** panel and expand **Voltage Area** to enable **Guardband** visibility as well.

You should find that there are two voltage areas displayed. **PD_RISC_CORE** in the top-right corner, and **DEFAULT_VA** for the remaining area.

You should also see that the macros in **DEFAULT_VA** have been placed (although not very carefully), but not the macros in **PD_RISC_CORE**: There are four macros inside that voltage area are placed on top of one another. This is normal. Only the top-level macros are placed at this time, macros belonging to non-default voltage areas will be placed during the next *Task 4*.

Lab 2

Task 4. Macro and Standard Cell Placement

1. From the Tasks palette, select **Cell Placement** → **Cell Placement**.

Note:

Note: There is a **Cell Placement** → **Hard Macro Constraints** entry that can be used to apply macro *keepout margins*. We are skipping this, for simplicity.

In the *Cell Placement* dialog, check the box next to **Use floorplanning placement**.

2. Run placement by clicking on **Apply**.
3. Enable pin visibility and examine the placement of the macros in the GUI.

You should see that the placer has automatically created channels between the macros and has added soft placement blockages in the narrow channels, to reduce congestion and improve routability. The more pins along a macro's edge, the larger the channel width.

You should also see that the macros have been flipped so that the sides with more pins face each other. This is done to minimize the overall channel area between macros.

Task 5. Place the Block Ports

1. You could also use the *Task Assistant* to place the pins of the block, but this time just select/run the corresponding commands from `run.tcl`:

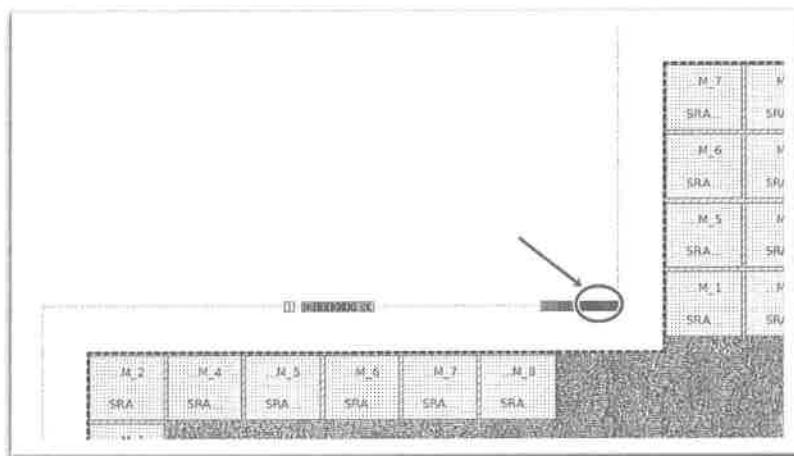
```
set_block_pin_constraints -self -allowed_layers {M3 M4 M5 M6}
place_pins -self
```

2. Have a look at the layout, and you should see that all the block's ports, the logical representations of the physical pins, have been placed.
3. Zoom in to individual ports to verify that the pins have, indeed, only been placed on layers M3-M6.

Note:

To be able to select the physical block "pins", turn on **Terminal** selectability (*terminals* are the physical pins of the current block).

4. Search for the *clk ports/terminals and zoom into their location. See run.tcl for hints.
5. Apply the commands from run.tcl to create a pin guide. This will constrain all the specified clock pins to be placed in a certain area. Afterwards, turn on pin guide visibility by turning on **Guide→Pin Guide** from the **View Settings → Objects** panel, or apply the command from the script. To see the pin guide, zoom in to the highlighted area as shown below.



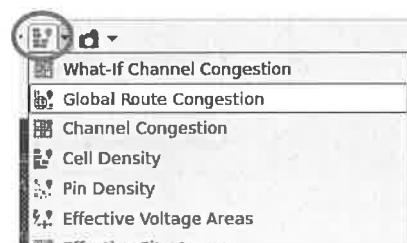
6. Change the width of the *clk ports (the terminal shapes) to 0.1 and the length to 0.4, then rerun pin placement. You should find that all clock pins are now located inside the pin guide, and that they have been resized as specified.

This was just one example of applying pin constraints. Review the man page to see what other adjustments are possible.

Task 6. Congestion Map

Now that macros and standard cells have been placed, as well as the block ports, it is a good idea to check if there are any congestion issues.

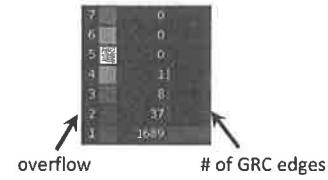
1. Bring up the congestion map by selecting the “*Global Route Congestion*” entry from the *maps* pull-down menu.
2. Select “**Reload**”, then confirm the dialog that appears by pressing **OK**.



Lab 2

3. After global routing completes, you should see that the layout has been updated to display the heat map.

You will not see any congestion to speak of: From the colored histogram, most “overflows” are 1, and very few overflows are greater than 2.



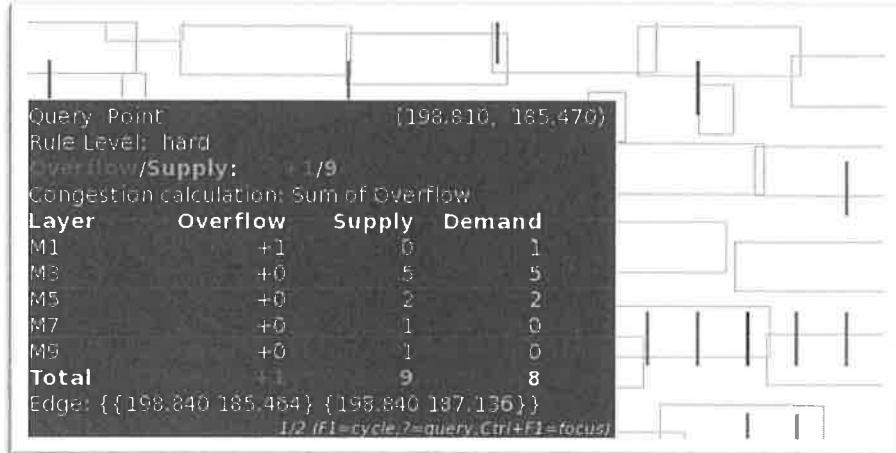
4. To make the display a little more interesting, try the following steps.

Change the **Bins/From/To** settings as shown in the following screenshot:

Bins: 7
From: -2 To: 2
Edges: Horizontal Vertical

Bins has been reduced from 9, and from/to has been changed from 0/7: Now, overflows from -2 to 2 have their own individual bin; All overflows greater than 2 are grouped into a combined bin, and the same for overflows less than -2.

This should drastically change what you see, by making the low overflows brighter/hotter: You will see that there are a lot of areas with 0 overflow. Display changes do not alter the results: The design has no serious congestion issues. If you want to see a detailed calculation of the overflow for an edge, move the mouse over that edge and you will see a popup with all the details, as shown here:

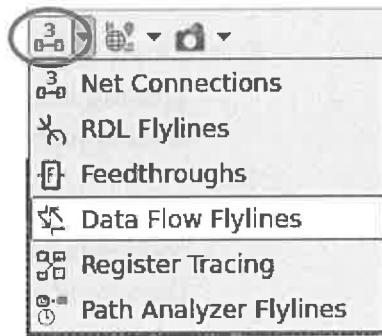


5. Close the congestion map, either by clicking on the *Draw Global Route Congestion ...* icon in the top banner, or by closing the palette.

Task 7. Analyze Macro Placement using DFF

Data Flow Flylines (DFF) is a feature that allows you to analyze not just simple pin-to-pin connections, but connections that cross through combinational gates as well as registers. This enables a very insightful view of your overall design and makes it easier to make informed decisions about macro placement which has a large impact on standard cell placement.

1. Bring up *Data Flow Lines* by selecting the appropriate entry from the *connectivity* pull-down, located to the left of the *maps* pull-down.
2. Select **Reload**. In the dialog that appears you can configure the tracing behavior of *DFF*. For example, you can specify how many register levels to trace through. The higher this number, the longer the calculations will take. For our purposes, accept the defaults and click on **OK**.
3. Once this completes, select **Macros** and **Ports** for items to **Include** in the flyline analysis, and click on **Apply** (highlighted in blue once you make changes) to invoke *DFF*. You can also change the configuration of this panel and choose “**Auto apply**”, so the changes take effect immediately.
4. Now select one macro to see its connections to other macros and ports. **Limit** the **tracing** by checking “**Number of registers**” or “**Number of gates**” and changing the **Min/Max** numbers. Using this method, you can quickly figure out whether objects are connected directly, or through several gate levels, or through several register levels. You can select several macros at the same time using the *Control* key. If you click on a flyline, you will get detailed information about the connection(s).
5. Note that *DFF* will not show you any flylines if they terminate at a register. You will only see flylines between macros or between macros and ports, depending on the selection you have made under “**Include**”. In the next *Task* we will show you how to perform *register tracing*.
6. Close the *DFF* palette.



Lab 2

Task 8. Register Tracing

1. Select *Register Tracing* from the *connectivity* pull-down, located right under the *Data Flow Flylines* entry.
2. Select any macro. The registers connecting to that macro will be highlighted immediately, skipping any logic that may be in between. Select *Show flylines* to see the flylines between the macro and registers.
3. To see the next level of registers, increase **Max levels** to **2** under the **Limit Tracing** heading, and under **Show Levels**, check **Level 2**. If there is a second level of registers they will be highlighted in a different color.
4. Under *Highlight*, you have the option to also display *End points*: These are the endpoints from the last level of registers displayed. You can also display *Direct end points*, which are endpoints that connect directly (level 0) to the selected source, i.e. our macro.
5. Close the *Register Tracing* palette.
6. We will assume for this lab that the macros are placed to our satisfaction, so the next step is to fix their location. You can do this by selecting the macros then clicking on  , or by entering:

```
set_fixed_objects [get_flat_cells -filter "is_hard_macro"]
```

Task 9. PG Prototyping

PG prototyping can help you create a basic PG mesh very quickly. All you need to specify are the PG net names, the layers and the percentage of the layers you want to use for PG routing. This is most commonly used as a place-holder for the final power mesh, in order to check congestion issues that a PG mesh may cause.

1. From the *Tasks* palette, select **PG Planning → PG Prototyping**
2. Click the **Default** button in the PG Prototyping dialog to show the default settings. Notice that, by default, the upper-two metal **PG layers** from the technology file, MRDL and M9, are listed to be used as the vertical and horizontal PG layers, respectively.
3. Specify M8 and M7 instead of the default vertical and horizontal layers, respectively, using the pull-down menus.
4. Click on **Apply**. In a couple of seconds, you should see a basic PG mesh inserted, which does not route over hard macros.
5. You can remove the PG mesh by clicking on **Remove PG Routes**.
6. If you like, play around with the layer and percentage parameters to test different PG mesh configurations. Remove PG routes before re-applying new parameters.

In the next task you will build the final mesh using a pre-written script.

Lab 2

Task 10. Power Network Synthesis

Planning an entire power network for a design can be a complex task, especially in a multi-voltage environment. The purpose of this task is to just give you an idea of what is possible in *IC Compiler II* using pattern-based power network synthesis (PPNS), and to demonstrate how few commands it takes to create a full multi-voltage PG mesh.

1. Quickly review the script that inserts the entire power structure. Open the file `scripts/pns.tcl` in an editor (or in the script editor!).

After first deleting any pre-existing PG mesh structures, you will see how the patterns are created (`create_pg_*_pattern`), followed by the strategies (`set_pg_strategy`). Once both are defined, the strategies are implemented using `compile_pg`.

2. Source the script:

```
source scripts/pns.tcl
```

3. Once the script has completed, review the power mesh, the macro PG connections, and the standard cell rails in the layout view.

Note: The power mesh will have a few issues here and there, which will have to be taken care of for final implementation.

4. Save and Exit out of IC Compiler II:

```
save_lib  
exit
```

You have completed the floorplanning lab!



3

Placement and Optimization

Learning Objectives

The purpose of this lab is to familiarize you with the placement capabilities in *IC Compiler II*.

You will perform placement on the ORCA_TOP design, after applying a number of settings. Reports will be generated to monitor and track the progress of the design.

After completing this lab, you should be able to:

- Check the readiness of the design for placement
- Apply settings for place_opt
- Perform placement and optimization
- Analyze timing and design quality of results (*QoR*)



Lab Duration:
60 minutes

Lab 3

Instructions

Answers / Solutions

You are **encouraged** to refer to the back of the lab to verify your answers, or to obtain help with the execution of some steps.

Task 1. Load and Check the Initial Design

1. Change to the work directory for the placement lab, then start ICC II in GUI mode and load the design:

```
UNIX% cd lab3_place  
UNIX% icc2_shell -gui -f load.tcl
```

The script copies the block that completed *design setup*, called ORCA_TOP/init_design, to a block called ORCA_TOP/place_opt, and opens the copied block.

2. Generate a timing QoR summary.

```
report_qor -summary
```

As to be expected, you should see a high WNS/TNS; no placement or any optimizations have been performed yet.

Task 2. Place and Optimize the Design

To support a more immersive and interesting lab experience, ***there are no step-by-step instructions for this lab.***

Instead, you are asked to open the file **run.tcl** in the *ICC2 script editor* and exercise the commands line by line. We are specifically asking you not to just source the entire file, as this would defeat the purpose which is to understand how all the options and commands play together. If there is an option that does not make sense, have a look at its *man* page.

The following sections provide additional information and comments. The sections are ordered in such a way that you can refer to them as you go through the script.

If you like, you can diverge from the commands in **run.tcl**, or you could try different efforts, different settings etc. Note though that the runtimes might vary. Make sure you only run the **place_opt** command once, as the runtime is relatively high.

The following section is designed to be a guide through the lab. It contains information on the items that must be configured and run, and some additional questions.

If you need help, talk to your instructor.

Pre-placement Checks

Before performing placement and optimization, it is best to ensure that there are no issues that will prevent `place_opt` from doing its job.

Question 1. Are there any remaining ideal nets?

.....

Question 2. What is the maximum routing layer set for the block?

.....

The design contains scan chains that are specified using *SCANDEF* as part of the data setup step.

Question 3. How many scan chains exist in the design?

.....

It can be important to establish whether the design has any high-fanout nets, or nets with a certain fanout. In ICC II, this is analyzed using `report_net_fanout`. Use the commands shown in `run.tcl` to answer the following question:

Question 4. How many non-clock high fanout nets exist with a fanout larger than 60?

.....

Pre-placement Settings

For technologies of 12 nm and below, you should use the `set_technology` command to configure ICCII. This command changes several application options to support the given technology.

To insert tie cells during `place_opt`, library tie cells must not have the `dont_touch` attribute applied, and they must be included for ‘optimization purpose’.

Question 5. What command/option is used to include cells for optimization?

.....

Lab 3

Leakage + Dynamic Power Optimization Settings

Leakage and dynamic power optimization are not performed by default during `place_opt`. If important, enable the appropriate power optimization (*leakage*, *dynamic* or *total*) and make sure that at least one scenario is enabled for leakage and/or dynamic power, if applicable.

Application option: `opt.power.mode`

Layer Optimization / Route Driven Extraction (RDE)

Layer optimization identifies long and timing critical nets, then promotes them to higher, low-resistance metal layers. The assigned min/max layer constraints from `place_opt` are carried all the way through to post-route optimization.

Application option: `place_opt.flow.optimize_layers`

Alternatively, with *route-driven extraction*, global routing is run on the initially placed design to construct an RDE extraction table. This extraction is used subsequently for all virtual net extraction which is then used for all pre-route optimizations (`place_opt` and `clock_opt`). This improves the pre- to post-route timing correlation. RDE is on-by-default for 16nm and below technologies (setting: `auto`). You can explicitly turn this on by setting this to “`true`”.

Application option: `opt.common.enable_rde`

ICG Optimization

ICG optimization is useful only if a design is known to have problems meeting ICG enable setup timing which is not the case in our design.

In the interest of reasonable run times, it is recommended not to enable this optimization for this lab.

Application option: `place_opt.flow.optimize_icgs`

Placement and Analysis

Make sure the scan chains are optimized during the `place_opt` run. Verify that the DFT optimization option is set appropriately.

Question 6. What is the app option for optimizing the scan chains and what is its default setting?

.....
.....

The advanced legalizer is generally recommended for 12nm technologies and below. The lab is based on 32nm technology, but if you want to enable the advanced legalizer anyway, set `place.legalize.enable_advanced_legalizer` to true.

Run `place_opt` (you may choose to run each stage individually – see the `run.tcl` file). If you execute all five stages in a single run, this may take around 10 minutes to complete, so take a break.

After `place_opt` completes, check the design for congestion and timing issues.

You should find that there is no congestion to speak of. Turn on visibility of keepout margins (**View Settings→Cell→Keepout Margin**) and you will see that all the channels between RAMs were blocked. This is an important contributor to the good congestion situation, as well as the good runtime and timing.

Timing should be met as well, especially if you have enabled the advanced legalizer.

You have completed the placement lab.



Answers / Solutions

Question 1. Are there any remaining ideal nets?

The `report_ideal_network` reports no ideal networks for all scenarios. If there are remaining ideal nets, you would have to remove them using the `remove_ideal_net` command.

Question 2. What is the maximum routing layer set for the block?

The `report_ignored_layers` command indicates that the maximum layer is set to **M8**.

Although the technology being used has 9 metal layers, we limit signal routing to metal 1 through 8 (M1 – M8). Metal 7 and 8 are used also for the power mesh, which limits the available resources. Metal 9 is not used in this design.

Question 3. How many scan chains exist in the design?

8. You can see this information near the end of the command `report_design -summary`, or you can use `get_scan_chain_count`.

Question 4. How many non-clock high fanout nets exist with a fanout larger than 60?

There are 10 total nets with a fanout ≥ 60 . By eliminating any *net name* containing “clk”, or *fanout driver pin* containing “CLK” you should find that there are 2 non-clock nets with a fanout ≥ 60 .

You can also use the more precise method, shown in `run.tcl`, using “**-filter net_type==signal**”.

Question 5. What command/option is used to include cells for optimization?

`set_lib_cell_purpose -include optimization`

Question 6. What is the app option for optimizing the scan chains and what is its default setting?

It is `opt.dft.optimize_scan_chain` and the default is true

5

Design Setup

Learning Objectives

The purpose of this lab is to familiarize you with *design setup* in IC Compiler II.

After completing this lab, you should be able to:

- Create an NDM design library
- Load the netlist, and apply the UPF
- Load floorplan- and scan-DEF files
- Confirm placement site and routing layer settings
- Debug some common design setup mistakes



Lab Duration:
40 minutes

Lab 5

Introduction

The *design setup* task is the most important step to perform correctly. Faulty setup can lead to many problems downstream, which can have a large negative impact on the design schedule. Once *design setup* is completed, you rarely need to revisit this task (unless the design or constraints change) and can focus on the more productive tasks like placement, CTS and routing.

The design setup steps that you will be performing include:

- Creating an NDM design library
- Loading the netlist, floorplan- and scan-DEF files
- Loading multi-voltage design data: UPF and voltage area definition
- Applying/confirming placement site and routing layer settings

Relevant Files and Directories

All files for this lab are located in the *lab56_setup* directory under your home directory.

lab56_setup/

run5.tcl Script with all the commands executed in this lab.

setup.tcl Setup variables and general settings.

.synopsys_icc2.setup Read by *IC Compiler II* upon startup.

ORCA_TOP_design_data/

ORCA_TOP.v Verilog gate level netlist.

ORCA_TOP.upf UPF multi-voltage design intent.

ORCA_TOP.fp/ Floorplan directory written by ICC II `write_floorplan`.

ORCA_TOP.scandef SCAN-DEF scan-chain definition.

../ref/CLIBs Directory containing the SAED32nm cell libraries

Lab 5

Instructions

Answers / Solutions

You are encouraged to refer to the end of this lab to verify your answers.

Task 1. Directory Structure and Invoking *ICC II*

1. Change your current directory to **lab56_setup** and look at the contents of the directory:

```
UNIX% cd lab56_setup  
UNIX% ls -al
```

This directory is shared with the next lab, in which you will complete the setup process (timing setup).

The **rm_setup/** and **ORCA_constraints/** directories, as well as the **run6.tcl** file are not listed on the previous page, because they are not used in this lab.

2. Invoke IC Compiler II:

```
UNIX% icc2_shell -gui
```

3. Open the **run5.tcl** file in the *ICC II script editor*, as in previous labs.

This file contains all the commands that you will be executing in this lab. Select the commands from this file and use **Run Selection**, instead of typing them yourself, to save time and avoid typing errors.

4. Open another **terminal window** and change your current directory to **lab56_setup**. You will use this window to examine the files that will be used in this lab.

Task 2. Setup Variables and Settings

1. In the last *terminal* window that you just opened, view the **setup.tcl** file in the **lab56_setup** directory.

Question 1. What is the default value of the `search_path` application variable?

(HINT:

```
printvar sear[TAB] or  
echo $search_path or  
get_app_var search_path or  
report_app_var search_path )
```

.....

Question 2. Which commands in `run5.tcl` will use the `search_path` application variable to locate their specified file(s)?
(list just the commands, without their options and arguments)

.....

.....

Question 3. Which command in `run5.tcl` uses the user-defined `TECH_LIB` and `REFERENCE_LIBRARY` variables?

.....

Question 4. From `setup.tcl`: How many reference libraries (labeled “saed32”) are being used?

.....

Question 5. From `setup.tcl`: Up to how many cores are enabled for multi-threading?

.....

Lab 5

2. By looking at the **SEE ALSO** section at the bottom of the *man* page of the `set_host_options` command, use the appropriate command to answer the next question (this is a useful technique to find *related* commands):

Question 6. How many cores are enabled, by default?

.....

The `suppress_message` commands at the bottom of `setup.tcl` are useful for suppressing *information* and *warning* messages that are expected, and either not useful or known to be benign. This helps to de-clutter the run log, making it easier to locate warnings and errors of real concern, when running scripts in batch mode.

3. Close the `setup.tcl` file – **do not save it**.
4. From the `run5.tcl` file in the ICC II GUI script editor, select and run the command to **source** the `setup.tcl` file:

```
source -echo setup.tcl
```

5. Confirm the applied settings:

```
printvar search_path  
printvar REFERENCE_LIBRARY  
report_host_options  
print_suppressed_messages
```

You might notice that there are additional suppressed messages beyond the ones listed in the setup file (CTS-725, POW-001, ...). These are tool-defaults, and this is usually done in order to reduce output verbosity. To see these messages, use `unsuppress_message`.

Task 3. Design Library, Netlist and UPF

1. Create the design library:

```
create_lib \
    -use_technology_lib $TECH_LIB      \
    -ref_libs           $REFERENCE_LIBRARY \
    ORCA_TOP.dlib
```

You will get the following error message: **Error: technology library '.../ref/CLIBs/saed32_1p9m_tech.ndm' does not match any library on the reference library list. (LIB-059)**

2. **Correct the problem** in the `setup.tcl` file, then **repeat the necessary commands** until a design library is successfully created.

Question 7. What is the name of the newly-created design library?

.....

Question 8. Did the design library show up in the `lab56_setup` directory?

.....

3. Read the *Verilog* netlist:

```
read_verilog -top ORCA_TOP ORCA_TOP.v
```

You will see another error message:

Error: File 'ORCA_TOP.v' cannot be found using search_path of: '. scripts ORCA_TOP_constraints'.
(FILE-002)

4. Using the file and directory structure shown on page 3, **correct the problem**, then **repeat the necessary commands** until the netlist is read without errors.
5. If, as in our case, the GUI is running, explicitly *linking* the block is not needed, since it is done automatically. You can, therefore, **skip the following command**. Linking ensures that all instantiated references can be found.

```
link_block
```

Lab 5

You will see the following warning messages:

```
Warning: Unable to resolve reference to 'SRAMLP2RW32x4'  
first referenced from module 'PCI_TOP'. (LNK-005)
```

```
Warning: Unable to resolve reference to 'SRAMLP2RW64x8'  
first referenced from module 'CONTEXT_MEM'. (LNK-005)
```

...

A linking problem occurs when the list of reference library pointers attached to the design library (specified by `create_lib -ref_libs ...`), is missing the library that contains one or more reference cells that are instantiated in the netlist.

6. The ORCA_TOP design uses four different SRAMs, which are all unresolved. If you look inside the directory that contains the reference cell libraries, you should be able to determine the missing reference library name.

Question 9. What is the name and location of the reference library containing the unresolved references?

-
7. The problem can be corrected in one of two ways:
 - Method #1: Use `set_ref_libs` to add the missing reference library to the design library.
 - Method #2: Correct the `setup.tcl` file and repeat the design setup steps.

The first method requires fewer steps, but, it does not correct the problem in the key design setup file, `setup.tcl`. If design setup needs to be repeated in the future (the netlist or the constraints are updated, for example), you will run into the same error.

The second method, which requires a few more steps, ensures that if design setup needs to be repeated, you will not encounter the same error.

You will execute both methods to fix the problem.

8. **Method #1:**

Add the missing reference library to the existing list using `set_ref_libs`, then re-link:

```
set_ref_libs -add ../ref/CLIBs/saed32_sram_lp.ndm  
link_block -force
```

The design should *link* without any warnings. While you *could* continue with the remaining design setup steps, you will first implement the second method to fix the problem at its source, the `setup.tcl` file.

9. Method #2:

First **edit `setup.tcl`** and add the missing reference library to the `REFERENCE_LIBRARY` list, then **close the design library**, then **repeat the main design setup steps**:

```
close_lib
source -echo setup.tcl
create_lib -use_technology_lib $TECH_LIB \
            -ref_libs $REFERENCE_LIBRARY ORCA_TOP.dlib
read_verilog -top ORCA_TOP ORCA_TOP.v
link_block
```

10. For final confirmation, run the command `report_ref_libs`.
11. You should see a *BlockWindow* in *IC Compiler II*'s GUI, which contains the physical or layout representation of the `ORCA_TOP` block. The layout contains all the standard cell and macro instances of the netlist, stacked on top of each other, in the lower-left corner. The blue-green rectangles are the hard macros, and the small purple rectangles in the lower-left corner are the standard cells. The block's I/O ports are also stacked on top of each other, and show up as a small light-blue square with a Greek-like Phi Φ symbol (actually an O and an I superimposed), just outside of the lower-left corner of the stacked cells.

Question 10. What is the name (block handle) of the newly-created current *block* in the design library?

Note:

The default block *handle* format is
`libName:blockName.viewName`.
The default *viewName* is *design*. A block can be saved with an optional *label*, in which case the block handle is:
`libName:blockName/labelName.viewName`. You will save the block with a label name at the end of the design setup steps.

Lab 5

12. If a design library contains multiple blocks with the same `blockName` (but with a different `labelName` and/or `viewName`), the block must be referred to by its unique block handle. If the design library contains only one block with a certain `blockName`, then that block can be referred to simply by its `blockName`. You can optionally include the `libName`, and/or `labelName`, and/or `viewName`, as demonstrated by executing these commands:

```
get_blocks ORCA_TOP
get_blocks ORCA_TOP.design
get_blocks ORCA_TOP.dlib:ORCA_TOP
get_blocks ORCA_TOP.dlib:ORCA_TOP.design
```

Note: Since this library contains only one block called `ORCA_TOP`, all of the above commands are accepted by *IC Compiler II*, and they all return the same full block handle.

13. Take a quick look at the UPF file located at `ORCA_TOP_design_data/ORCA_TOP.upf`. This UPF files defines:
 - Three power supply nets/ports: `VSS`, `VDD` and `VDDH`
 - Two power domains: `PD_ORCA_TOP` (the top level of the block) and `PD_RISC_CORE` (contains the `RISC_CORE` sub-design)
 - Level shifters for inputs and outputs of `PD_RISC_CORE`
 - Defines the power states of the power nets. Here, the power nets only define an ON-state
14. Close the UPF file – do not save it, then **load** and **commit** UPF:

```
load_upf ORCA_TOP.upf
commit_upf
```

Task 4. Load Floorplan and Scan-DEF

1. Load the floorplan generated by ICC II floorplanning:

```
source ORCA_TOP.fp/floorplan.tcl
```

Note: The ORCA_TOP.fp directory is located under the ORCA_TOP_design_data directory.

2. Look at the GUI *BlockWindow*, and you will see the complete mirrored-L shape floorplan of the ORCA_TOP block.
3. **Zoom in** and notice the complex P/G mesh structure, then zoom back out to a **full-zoom**. Since the P/G mesh makes it difficult to see the underlying structures clearly, we will turn off their visibility next.
4. In the *View Settings* panel, make the following changes, to improve the visibility of the floorplan:
 - **Route** → **Net Type** → **Power and Ground** → Uncheck visibility
 - **Terminal** → Check selectability
 - **Port** → Uncheck visibility
 - **Voltage Area** → Check visibility
5. Under the **Settings**→**View tab**, enable **Label settings**→**Scale fonts**. This improves the readability of the labels.
6. You can now more clearly see that:
 - All the macros are placed
 - Terminals (metal connection shapes) for the I/O and P/G ports are placed around the block boundary
 - A voltage area, called **DEFAULT_VA**, is defined for the entire core area (dashed purple outline)
 - A second voltage area called **PD_RISC_CORE** is defined in the lower-right
 - Standard cells are still stacked on top of each other in the lower-left corner

Lab 5

7. Read the SCAN-DEF file:

```
read_def ORCA_TOP.scandef
```

Question 11. How many scan chains does the design have?

.....

8. Connect the P/G pins to the supply nets, and verify that there are no P/G connection errors:

```
connect_pg_net  
check_mv_design
```

Task 5. Placement Site and Routing Layer Settings

1. Confirm that the placement site called `unit` is set as the *default* site definition (its `is_default` attribute should be `true`):

```
get_attribute [get_site_defs unit] is_default
```

2. Confirm that *Y-symmetry* is applied to the standard cells:

```
get_attribute [get_site_defs unit] symmetry
```

In our case, this was defined while preparing the *technology-only library*, which you learned about in the previous unit. If this was not done, or you did not use a technology-only NDM, you would have to apply this command:

```
set_attribute [get_site_defs unit] symmetry {Y}
```

Note: *You do not need to run the above command!*

Question 12. Does Y-symmetry mean that standard cell can be flipped in the Y-direction (along the X-axis), or flipped in the X-direction (along the Y-axis)?

.....

3. Confirm that the metal layer preferred *routing directions* are defined:

```
get_attribute [get_layers M?] routing_direction
```

As noted above, this information was specified when preparing the technology-only NDM library.

4. Confirm that all metal layers are available for signal routing (none are *ignored*), by default, then limit routing to M6 or lower:

```
report_ignored_layers
set_ignored_layers -max_routing_layer M6
report_ignored_layers
```

Lab 5

Task 6. Save the Block

You have completed all the *design setup* steps; *Timing setup* will be performed in the next unit. This is a good time to save the block.

1. List the files and directories in the current working directory (CWD), `lab56_setup`:

```
ls -l
```

Question 13. Does the `ORCA_TOP.dlib` design library exist in the current working directory?

-
2. Execute the following commands to rename the block with a *label*, `init_design`, which helps to identify it (as this version that has been *initialized*), and then to save the block and library:

```
rename_block -to_block ORCA_TOP/init_design  
save_block -or- save_lib
```

Question 14. Does the `ORCA_TOP.dlib` design library show up now?

-
3. Exit out of *IC Compiler II*. The GUI will ask for confirmation – click **OK**:

```
exit
```

You have completed the Design Setup lab!



Answers / Solutions

Question 1. What is the default value of the `search_path` application variable?

(HINT:

```
printvar sear[TAB] or  
echo $search_path or  
get_app_var search_path or  
report_app_var search_path )
```

The current directory: .

Question 2. Which commands in `run5.tcl` will use the `search_path` application variable to locate their specified file(s)?
(list just the commands, without their options and arguments)?

`source, read_verilog, load_upf, read_def.`

(Since the `$TECH_LIB` and `$REFERENCE_LIBRARY` variables include the path to each library, the `create_lib` command will not use the `search_path` variable)

Question 3. Which command in `run5.tcl` uses the user-defined `TECH_LIB` and `REFERENCE_LIBRARY` variables?

`create_lib`

Question 4. From `setup.tcl`: How many cell reference libraries are being used?

3: HVth, LVth and RVth cell libraries (containing standard cells and level shifters).

Question 5. From `setup.tcl`: Up to how many cores are enabled for multi-threading?

The code lines in the script will reserve as many threads as available on the machine, but no more than 8.

Question 6. How many cores are enabled, by default?

From `report_host_options`: `max_cores: 1`

Task 3. Design Library, Netlist and UPF**2. Correct the problem ...**

Add the technology library to the top of the `REFERENCE_LIBRARY` list in the `setup.tcl` file, then `source` the file:

```
set REFERENCE_LIBRARY [join "
    $TECH_LIB
    ./ref/CLIBs/saed32_hvt.ndm
    ./ref/CLIBs/saed32_lvt.ndm
    ./ref/CLIBs/saed32_rvt.ndm
"]
```

Question 7. What is the name of the newly-created design library?

ORCA_TOP.dlib: As echoed in the run log, or, from the `create_lib` argument without an option, or, by typing `current_lib`

Question 8. Did the design library show up in the `lab56_setup` directory?

No. The design library will not appear (as a UNIX directory) until the *block* is saved, which will be done near the end of the design setup steps.

4. Using the file and directory structure shown on page 3, correct the problem

...

Add the `ORCA_TOP_design` directory, which contains the *Verilog* netlist (and other) design files, to the `search_path` in `setup.tcl`, then source the file:

```
set_app_var search_path ". scripts
    ORCA_TOP_constraints ORCA_TOP_design_data"
```

Question 9. What is the name and location of the reference library containing the unresolved references?

`./ref/CLIBs/saed32_sram_lp.ndm`

9. Method #2:

First close the design library, then **edit setup.tcl** and add the missing reference library to the **REFERENCE_LIBRARY** list ...

```
set REFERENCE_LIBRARY [join "
$TECH_LIB
../ref/CLIBs/saed32_hvt.ndm
../ref/CLIBs/saed32_lvt.ndm
../ref/CLIBs/saed32_rvt.ndm
../ref/CLIBs/saed32_sram_lp.ndm
"]
```

Question 10. What is the name (block handle) of the newly-created current *block* in the design library?

From **current_block**:

ORCA_TOP.dlib:ORCA_TOP.design

Question 11. How many scan chains does the design have?

8

```
Number of Processed/Read DEF Constructs
-----
VERSION : 1/1
DIVIDERCHAR : 1/1
BUSBITCHARS : 1/1
DESIGN : 1/1
SCANCHAINS : 8/8
```

You can also get the scan chain count by running the command **get_scan_chain_count**.

Question 12. Does Y-symmetry mean that standard cell can be flipped in the Y-direction (along the X-axis), or flipped in the X-direction (along the Y-axis)?

Standard cell can be flipped in the **X-direction, along the Y-axis**.

Question 13. Does the ORCA_TOP.dlib design library exist in the current working directory?

No. It is still not there.

Question 14. Does the ORCA_TOP.dlib design library show up now?

Yes!

6

Timing Setup

Learning Objectives

The purpose of this lab is to familiarize you with *timing setup* in IC Compiler II.

After completing this lab, you should be able to:

- Perform MCMM setup:
 - Define corners, modes and scenarios required for analysis and optimization
 - Load MCMM timing constraints
- Confirm implementation phase readiness with various reports and checks, as well as a zero-interconnect timing sanity check



Lab Duration:
40 minutes

Lab 6

Introduction

The *timing setup* task completes the design setup, which is the most important step to perform correctly. As explained earlier, once *setup* is completed, you rarely need to revisit this task (unless the design or constraints change) and can focus on the more productive tasks like placement, CTS and routing.

The timing setup steps that you will be performing include:

- Performing MCMM setup, which includes defining the corners, modes and scenarios required for analysis and optimization, and loading the MCMM constraints
- Confirming implementation phase readiness with various reports and checks
- Performing a zero-interconnect timing sanity check

Relevant Files and Directories

All files for this lab are located in the *lab56_setup* directory under your home directory.

lab56_setup/	
run6.tcl	Script with all the commands executed in this lab.
setup.tcl	Setup variables and general settings.
ORCA_TOP_constraints/	
ORCA_TOP_c_*.tcl	Corner constraints.
ORCA_TOP_m_*.tcl	Mode constraints.
ORCA_TOP_s_*.tcl	Scenario constraints.
ORCA_TOP_port_lists.tcl	Port definitions used by constraint commands.
scripts/	
mcmc_ORCA_TOP.tcl	Script to define MCMM corners, modes, and scenarios, and load their respective timing constraints.

Instructions

Answers / Solutions

You are encouraged to refer to the end of this lab to verify your answers.

Task 1. Open the Block from Lab 5

This lab is a continuation of Lab 5. It expects that block ORCA_TOP/init_design exists, which is created at the end of Lab 5. If you did not complete Lab 5 yet, do that first.

Alternatively, to catch up, run: `icc2_shell -f .solution/complete5.tcl`

1. Invoke *IC Compiler II* from the `lab56_setup` directory:

```
UNIX% cd lab56_setup
UNIX% icc2_shell -gui
```

2. Open the `run6.tcl` file in the *ICC II script editor*, as in previous labs.

This file contains all the commands that you will be executing in this lab. Select the commands from this file and use **Run Selection**, instead of typing them yourself, to save time and avoid typing errors.

3. Source the `setup.tcl` file:

```
source -echo setup.tcl
```

4. Open the block: This can be accomplished either by first opening the library, and then the block, or, by using the full block handle, which includes the library, in which case you do not need to first open the design library:

```
open_block ORCA_TOP.dlib:ORCA_TOP/init_design
# OR
open_lib ORCA_TOP.dlib
open_block ORCA_TOP/init_design
```

Lab 6

Task 2. Multi-Corner Multi-Mode Setup

1. In a different *terminal* window, open the file
`scripts/mcmm_ORCA_TOP.tcl`.

This script first creates the *modes*, *corners* and *scenarios* needed for multi-corner multi-mode (MCMM) optimization of this design.

Note: The script takes advantage of Tcl *arrays*
(`set arrayName(varName) varValue`) to create
efficiently-coded *foreach* loops.

- Question 1.** What are the names of the *modes*, *corners* and *scenarios* that will be created?

Modes:

Corners:

.....

Scenarios:

.....

.....

2. The next section of the `mcmm_ORCA_TOP.tcl` script sources the mode-, corner- and scenario-specific constraints files into their respective newly-created modes, corners and scenarios.

The constraints files are located in `ORCA_TOP_constraints/`. If you have the time, **take a quick look** at one of each of the mode (`_m_`), corner (`_c_`) and scenario (`_s_`) files.

3. The last section of the `mcmm_ORCA_TOP.tcl` script configures the analysis types that are activated for the scenarios.

Use the `man` page for `set_scenario_status`, if needed, to help you answer the following questions:

- Question 2.** Which scenarios will be *active*?

.....

Question 3. Which analysis types will be enabled for the test.ss_125c scenario?

-
4. Close the mcmm_ORCA_TOP.tcl file – **do not save it**, then **source** it:

```
source -echo mcmm_ORCA_TOP.tcl
```

5. **Look** at the *log* messages that were generated in the *icc2_shell* window:

After scenario creation, the messages confirm that all scenarios are active for all analysis types:

```
Created scenario test.ff_125c for mode test and corner ff_125c  
All analysis types are activated.
```

The warning messages about the *virtual* and *generated clocks*, which occur when sourcing the mode constraints, are just informational, and can be ignored (*virtual clocks*, by definition, have no sources).

6. **Verify** that the active analysis types for the test.ss_125c scenario match your answer to the previous question.
7. **Ensure** that there are no *propagated clocks* prior to clock tree synthesis:

```
set cur_mode [current_mode]  
foreach_in_collection mode [all_modes] {  
    current_mode $mode  
    remove_propagated_clocks [all_clocks]  
    remove_propagated_clocks [get_ports]  
    remove_propagated_clocks [get_pins -hierarchical]  
}  
current_mode $cur_mode
```

8. **Verify** that the **current_mode** and **current_corner** are consistent with the **current_scenario**.
9. **Change** the **current_mode** or **current_corner** and notice that the **current_scenario** changes accordingly. Conversely, if you change the **current_scenario** this changes the **current_mode** and/or **current_corner**.

Note: There are no scenarios for test mode in *_m40c corners.

Lab 6

10. **Generate** a *mode report*, which creates a convenient table-format summary of the active and analysis type status of all scenarios, grouped by mode:

```
report_mode
```

Right after the name of the mode you will see this line:

Current: false Default: false Empty: false

Current refers to whether this mode is the current mode or not.

Default is only true if you didn't create any modes on your own, in that case ICC II would have single mode named *default*.

Empty is true if you have not applied any constraints to this mode.

11. **Generate** a *pvt report*, to find out if there are any mismatches between the PVT values defined in each corner, versus the available library PVTs:

```
view report_pvt
```

12. Look at the summary section for each of the four corners (between the horizontal dashed lines), and answer the following questions:

Question 4. Which corner(s) have PVT mismatches?

.....

Question 5. What is mismatching – process, voltage and/or temperature?

.....

13. The information below the warning summary section lists the details of each library that has a mismatch (based on the cells instantiated in the netlist). A quick way to find out what the problems are is to look at the lines with an asterisk or star (*).

Question 6. What is causing all of the mismatches?

.....

.....

.....

Based on the name of the corner with the mismatches (`ss_m40c`), as well as the name of the other corner with that same temperature (`ff_m40c`), it is reasonable to conclude that the problem is with the user-specified temperature constraint of -55, not with the characterized corner of the libraries.

14. **Exit** the PVT report.
15. **Make the necessary correction** in the appropriate constraints file located in the `ORCA_TOP_constraints` directory.
16. **Re-execute** the commands in **steps 4, 7 and 11** on the previous pages, until a clean PVT report is obtained.
17. **Save the block/library**

```
save_lib
```

18. Run the following command to list the blocks:

```
list_blocks
```

Question 7. Has the block been saved?

.....

Task 3. Zero-Interconnect (ZIC) Timing Sanity Check

It is a good idea to perform a *zero-interconnect (ZIC) timing sanity check*, to ensure that the netlist has a chance at meeting timing after the design is placed and routed.

1. Generate a *QoR summary* report, then set the design in ZIC timing mode and generate another *QoR summary* report:

```
report_qor -summary -include setup
set_app_options -list \
    {time.delay_calculation_style zero_interconnect}
report_qor -summary -include setup
```

You will notice a big difference in non-ZIC and ZIC timing. This is due to the long, estimated routes between the unplaced standard cells in the lower-left corner, and the hard macro cells in the block, as well as the I/O terminals

Lab 6

around the block. In ZIC timing, the RC parasitics of these long, estimated routes are set to zero which drastically improves *setup* timing.

This QoR report is very useful to get a high-level summary of the *worst negative slack* (WNS) timing, as well as the *total negative slack* (TNS), and the *number of violating endpoints* (NVE) for each scenario.

At first glance, when looking at the second, ZIC QoR report, it appears that the design has a serious problem! Two of the three setup timing scenarios have WNS violations of ~2.6 ns!

2. First let us make sure that these large violations are not due to unbuffered high fanout nets (HFNs) - assume a fanout of 50 or more is considered a HFN:

```
set_app_options -list \
    {time.high_fanout_net_pin_capacitance 0pF
     time.high_fanout_net_threshold 50}
update_timing -full
report_qor -summary -include setup
```

Since the results are the same, the violations are not caused by HFNs.

3. Notice that the WNS for the scenario in the -40 °C corner, func_ss_m40c, is not violating (has a positive WNS).

Question 8. Can you think of a reason why one scenario meets setup ZIC timing, while the others have a huge WNS violation?

.....
.....
.....

Next, you will investigate the large *setup* timing violations further, to confirm that optimization was, indeed, not done for these scenarios.

4. Generate a timing report for the five worst violating paths:

```
view report_timing -max_paths 5
```

5. In the view window, look at the incremental delay numbers in the column labeled **Incr**, for the first reported path. You should notice a couple of large ($> 1\text{ns}$) delays. If you look to the left of those large delays, at the name of the standard cell reference shown in parenthesis, you will find that they are small-sized (1x or 2x), high Vth gates (ending with X1_HVT or X2_HVT) which are the slowest cells available! Scroll down and look at the other four paths: You will find the same thing there. In fact, you should notice that these slowest cells are being used all along the timing-critical paths.

This is a clear indication that these paths were not optimized, which confirms that these scenarios were not considered during synthesis. Ideally, it would be best to re-synthesize the design under all key setup timing scenarios, which would provide a better starting netlist to IC Compiler II. This would result in a better initially-placed design, requiring less optimization, and thus better placement run-time. If re-synthesizing the design is not an option, there is still a good likelihood that optimization during the placement, CTS and routing phases will be able to eliminate, or drastically improve the timing in the other scenarios.

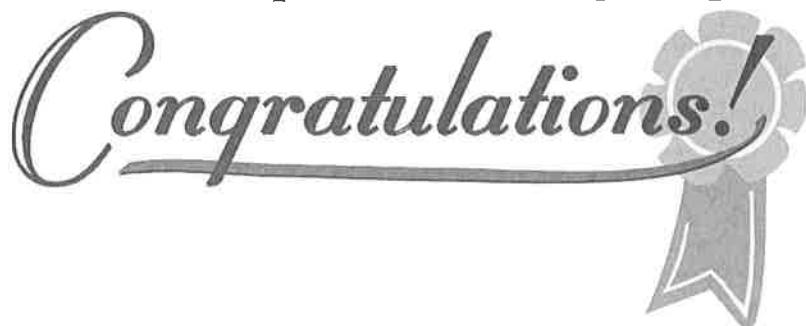
6. **Exit** the timing report view window.

You have completed all recommended timing setup steps!

7. **Exit** out of IC Compiler II:



You have completed the Timing Setup lab!



Answers / Solutions

Question 1. What are the names of the *modes*, *corners* and *scenarios* that will be created?

Modes: `func, test`

Corners: `ss_125c, ss_m40c, ff_125c, ff_m40c`

Scenarios: `func.ss_125c, func.ss_m40c,`
`func.ff_125c, func.ff_m40c, test.ss_125c,`
`test.ff_125c`

Question 2. Which scenarios will be *active*?

ALL scenarios. By default, all scenarios are *active* when created. The `-active false` option is used to de-activate scenarios, after-which `-active true` can be used to re-activate them.

Question 3. Which analysis types will be enabled for the `test.ss_125c` scenario?

Setup timing and logical DRCs (max transition, max capacitance, min capacitance). Leakage/dynamic power, hold timing as well as cell and signal EM analyses are disabled.

Question 4. Which corner(s) have PVT mismatches?

`ss_m40c`

Question 5. What is mismatching – process, voltage and/or temperature?

Temperature

Question 6. What is causing all of the mismatches?

The **effective** (library characterization) temperature of **-40** degrees Celsius does not match the user-**specified** value of **-55**.

Task 2. Multi-Corner Multi-Mode Setup

15. Make the necessary correction in the appropriate constraints file located in the **ORCA_TOP_constraints** directory.

In the **ORCA_TOP_c_ss_m40c.tcl** file make the following correction:

```
set_temperature -40
```

- Question 7.** Has the block been saved?

The date and time stamp show when the block was last saved to disk, which confirms that the block **has** been saved.

- Question 8.** Can you think of a reason why one scenario meets setup ZIC timing, while the others have a huge WNS violation?

One common reason is that synthesis was not performed in an MCMM environment. In our case, it was performed for a single mode and corner: The functional mode (`func`), and the slow-slow process at -40 °C corner (`ss_m40c`).

This page was left blank intentionally.

7

Running CTS

Learning Objectives

The purpose of this lab is for you to become familiar with the clock tree synthesis and Post-CTS optimization capabilities in *IC Compiler II*.

You will perform clock tree synthesis on the ORCA_TOP design using the recommended flow, 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:

- Apply basic settings for clock tree synthesis and data-path optimization
- Run either the classic CTS or the CCD flow
- Analyze CTS quality-of-results (QoR)



Lab Duration:
70 minutes

Lab 7

Instructions

Answers / Solutions

You are **encouraged** to refer to the back of the lab to verify your answers, and to look for hints.

Task 1. Load and Analyze the Placed Design

In this task, you will load the resulting design after placement and optimization and perform pre-CTS checks.

1. Change to the work directory for the CTS lab, then load the placed design:

```
UNIX% cd lab7_cts  
UNIX% icc2_shell -gui -f load.tcl
```

The script will make a copy of the `place_opt` block and open it.

2. Open the `run.tcl` file in the *ICC II script editor*, as in previous labs.

This file contains all the commands that you will be executing in this lab. Select the commands from this file and use **Run Selection**, instead of typing them yourself, to save time and avoid typing errors.

3. Generate a timing QoR summary.

```
report_qor -summary
```

Question 1. From a timing stand-point – is the design ready for CTS? What about the *hold* violations?

.....
.....

4. Set the *current mode* to “func”.
5. Generate a clock report:

```
report_clocks
```

Question 2. How many *master* clocks are defined?

.....

Question 3. How many *generated* clocks are defined?

.....

Question 4. What type of clock are the remaining clocks?

.....

Question 5. What is the source of the SD_DDR_CLK clock?

.....

6. Generate a *clock skew* report:

```
report_clocks -skew
```

Question 6. What is smallest/largest Setup Uncertainty?
What is smallest/largest Hold Uncertainty?

.....

.....

7. Generate a *clock groups* report:

```
report_clocks -groups
```

Question 7. Which clock groups are *mutually exclusive* or *asynchronous*?

.....

.....

.....

Lab 7

8. Generate a clock tree summary report for both *modes* (default report):

```
v report_clock_qor -modes func  
v report_clock_qor -modes test
```

- Question 8.** What is the big difference between the two modes?
(Hint: Look at the clock names)

-
Question 9. How many sinks does SD_DDR_CLK have?

.....

9. Generate a port report on the start point or *source* of SD_DDR_CLK (see Question 5):

```
report_ports [get_ports sd_CK]
```

- Question 10.** Why does SD_DDR_CLK have zero *sinks*?
(Hint: Look at the port direction)

.....

.....

Task 2. CTS Setup and Configuration

- Since CTS setup (clock tree balancing constraints, NDR rules, etc.) will be covered in the next unit, perform these setup steps by sourcing a file:

```
source scripts/cts_ex_ndr.tcl
```

- Ensure that the correct scenarios are enabled for hold fixing. To find all active scenarios that are enabled for hold:

```
get_scenarios -filter active&&hold
report_scenarios
```

Question 11. Are the scenarios configured properly for hold fixing?

.....

Complete the scenario setup for hold (look at run.tcl). Also, double check that all scenarios are active.

- You can control which buffers or delay cells should be used for hold fixing using `set_lib_cell_purpose`. Execute the three corresponding commands from the run.tcl file.
- If you like, you can increase the *effort* for maximum hold timing optimization, although it is not required for this design:

```
set_app_options -name clock_opt.hold.effort \
                 -value high
```

- In order to reduce scan chain hold timing violations, it is recommended to enable *scan-chain reordering* to minimize crossings between clock buffers:

```
set_app_options \
    -name opt.dft.clock_aware_scan_reorder \
    -value true
```

- Enable *clock reconvergence pessimism removal* by executing the corresponding line from run.tcl.

Lab 7

Task 3. Post-CTS I/O Latency

For our design, we do not want the I/O latencies to be updated, except for `v_PCI_CLK`. This clock is a *virtual* clock, so latency adjustment needs to be configured to update the clock:

```
foreach_in_collection mode [all_modes] {  
    current_mode $mode  
    set_latency_adjustment_options -exclude_clocks "*"  
    set_latency_adjustment_options \  
        -reference_clock PCI_CLK \  
        -clocks_to_update v_PCI_CLK  
}
```

Note: Generally, if the clock for the input/output constraints is the same as the internal clock, no configuration needs to be done – their I/O latencies will automatically be adjusted.

Task 4. Run Comprehensive Clock Tree Checking

- Now that you have analyzed the clocks using the manual methods discussed earlier, generate a *clock tree check* report to see what other potential problems might appear during CTS:

```
v check_clock_trees
```

You should see a long report with a “Summary” and a “Details” section.

The summary section will show you how many problems were found of each problem category, and if there is a suggested solution, for example:

CTS-019	2	None	Clocks propagate to output ports
CTS-905	4	None	There are clocks with no sinks

For more details, review the Details section: The detailed section for CTS-0905 (at the end of the report) complains about clocks without sinks, which you have already analyzed in earlier steps. Four clocks are listed, related to ports `SD_DDR_CLK` and `SD_DDR_CLKn` (repeated for each mode, `func` and `test`).

CTS-903 46 None Cells instantiated in the clock network are not in the clock reference list

CTS-904 12 None Some clock reference cells have no LEQ cell specified for resizing

These two warnings list cells that are used in the clock tree, and are either not enabled for the *CTS lib cell purpose*, or, they are enabled for the *CTS lib cell purpose*, but their “logically equivalent cells” (LEQs) are not. In either case this means that CTS will not be able to resize these cells (discussed in the next Unit).

CTS-009 12 None Cell instances in the clock tree have multiple conditional delay arcs between the same pins

CTS-013 35 None Cells in the clock network have a `dont_touch` constraint

This lists the cells that we have applied a `dont_touch` on.

For the purposes of this lab, all of these Warnings are acceptable, and can be ignored.

Classic CTS or CCD

The allotted lab time is such that you will most likely only have time for one of the CTS flows – choose either the classic CTS flow (Option A – Tasks 5 and 6, starting on the next page), or the CCD flow (Option B – Task 7, starting at page 12).

Note: For this design, the runtime for CCD is higher than for classic CTS.

If you are done early with one flow, try the other flow.

Lab 7

Task 5. Option A: Perform Classic CTS

Note: If you are performing this step after you have already performed option B (CCD), then you will need to re-load the design and re-apply all settings. To simplify this, just restart ICCII and use the script `scripts/load_all.tcl` – this script will re-load the design and set up everything up to this point.

1. Perform clock tree synthesis and route the clock trees:

```
clock_opt -to route_clock
```

The run should only take a few minutes. The above command will execute the first two stages: `build_clock` and `route_clock`.

2. Once the run has completed, review the CTS skew results. After looking at all results in all modes/corners, record results for the worst corner for the functional mode, `ss_125c`:

```
report_clock_qor
v report_clock_qor -type local_skew
report_clock_qor -type area
report_clock_qor -mode func -corner ss_125c \
-significant_digits 3
```

3. Record the various clock QoR statistics:

Number of clock buffers (clock repeaters):

Clock tree area (repeaters + other network cells):

Record the *global/local skew/max latency* numbers for the indicated clocks, in the slowest corner `ss_125c`:

ss_125c Corner	Global Skew	Local Skew	Max Latency
SYS_2x_CLK			
SDRAM_CLK			

4. Another very useful report is the *robustness* report:

```
v report_clock_qor -type robustness -mode func \
-corners ff_m40c -robustness_corner ss_125c
```

Multi-corner robustness is a measure of how the latency to each clock sink scales between the measured corner and a reference corner (specified with the `-robustness_corner` option). Every corner pair has a scaling factor, which is simply the ratio of the average latency in the measured corner over the average latency in the reference corner. The robustness value for an individual sink is the ratio of its latency in the measured corner over the latency in a reference corner, normalized against the scaling factor for those corners.

If a clock sink has a robustness value of 1, that means it exhibits typical scaling between the measured and reference corner. A robustness value greater than one indicates a sink has higher than average latency in the measured corner compared to the reference corner. Likewise, a robustness value less than one indicates a sink with less than average latency in the measured corner compared to the reference corner. The largest and smallest robustness value sinks have the worst multi-corner robustness, and could cause timing problems in the measured corner. If all clock sinks for a clock or skew group have a similar robustness value, then the clock tree is said to be *multi-corner robust*, and the skew can be maintained across those corners.

5. Generate a different skew report using the *clock timing* report: Concentrate on the `func` mode and the worst corner:

```
report_clock_timing -type skew -modes func \
-corners ss_125c -significant_digits 3
```

The reported Skew is the difference between the max and min Latency numbers, plus or minus the *clock reconvergence pessimism* (CRP).

Record Skew and maximum Latency for the indicated clocks:

	Skew	Latency (max)
SYS_2x_CLK		
SDRAM_CLK		

Lab 7

Note: Definition of the symbols on the right end of the report:

w	Worst-case operating condition
b	Best-case operating condition
r	Rising transition
f	Falling transition
p	Propagated clock to this pin
i	Clock inversion to this pin
-	Launching transition
+	Capturing transition

Question 12. Why are the skews reported by `report_clock_qor` and `report_clock_timing` different?

.....

.....

Task 6. Perform Post-CTS Optimization

In this task you will optimize the non-clock network logic to address any timing violations, and you will perform hold-fixing for the first time.

1. Generate a timing summary before optimization:

```
report_qor -summary
```

Note down the worst-case (Design) WNS/TNS/NVE numbers for setup and hold:

	WNS	TNS	NVE
Setup			
Hold			

2. Execute post-CTS optimization:

```
clock_opt -from final_opto
```

3. Once post-CTS optimization has completed, generate another timing summary.

Question 13. Are there any setup or hold violations left?

.....

4. Confirm that the design has no congestion issues.
5. Save the block and continue to **Task 8**.

Lab 7

Task 7. Option B: Concurrent Clock & Data Flow

In this task, you will use the *CCD* flow to build the clock trees and optimize the logic. **Note that *CCD* will take much longer to run compared to classic CTS.** If you have chosen option B right away, then continue with the first step.

Note: If you are performing this step after you have already performed option A, then you will need to re-load the design and re-apply all settings. To simplify this, just restart ICCII and use the script `scripts/load_all.tcl` – this script will re-load the design and set up everything to this point, ready for *CCD*.

1. Source the following script – this will make things a little more interesting for *CCD*, by **introducing a few setup timing violations, which will need to be fixed by the *CCD* algorithms**. Have a look at a timing summary afterwards:

```
source scripts/margins_for_ccd.tcl  
report_qor -summary
```

2. Note down the worst-case (Design) WNS/TNS/NVE numbers for setup and hold:

	WNS	TNS	NVE
Setup			
Hold			

3. Enable the *CCD* flow.
4. Run default `clock_opt`. This is the recommended way to run the *CCD* flow, all three stages (`build_clock`, `route_clock` and `final_opto`) will be executed. **This will run for at least 15 minutes.** Take a break.

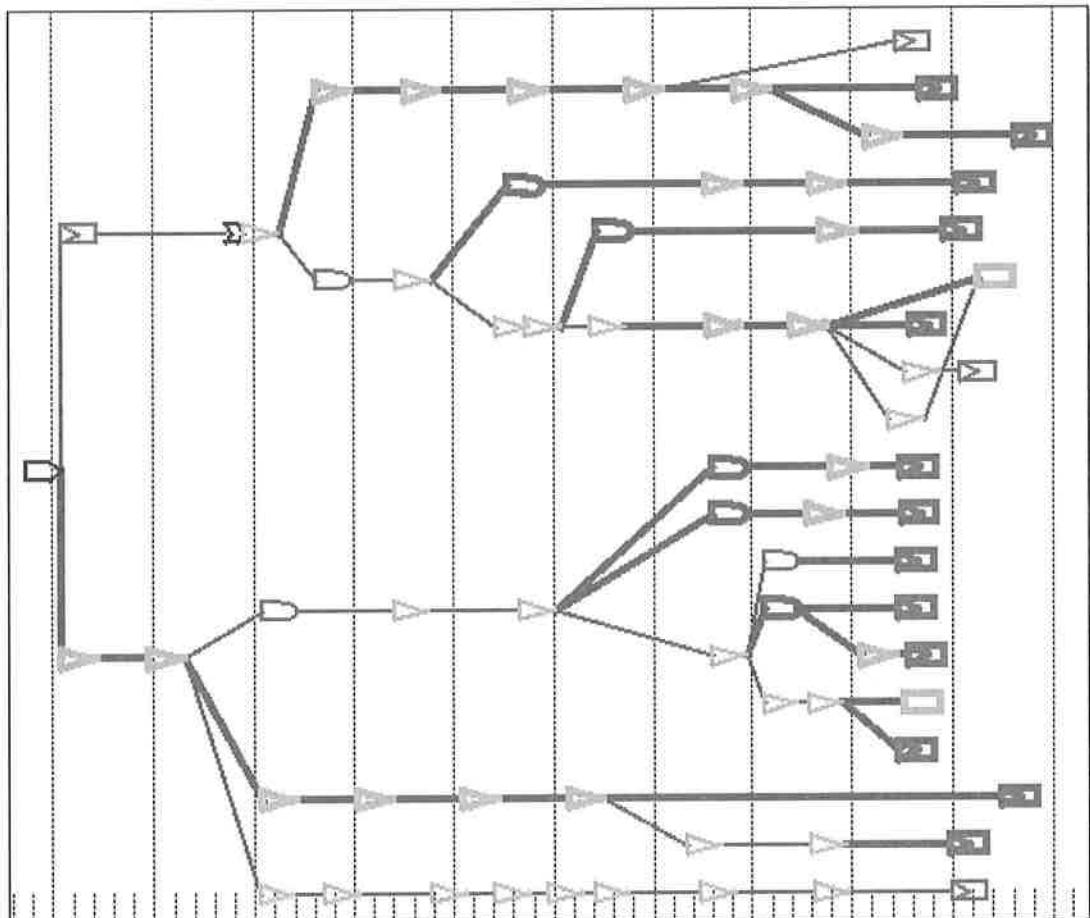
You may, of course, choose to run each stage by itself, and then perform intermediate analysis.

5. After completing the run, record the design QoR to see whether *CCD* was able to fix all the artificially introduced violations. Note: Do not compare results with the classic CTS! Timing was purposely made worse for *CCD*.
6. Save the block and **continue to Task 8**.

Task 8. Analysis

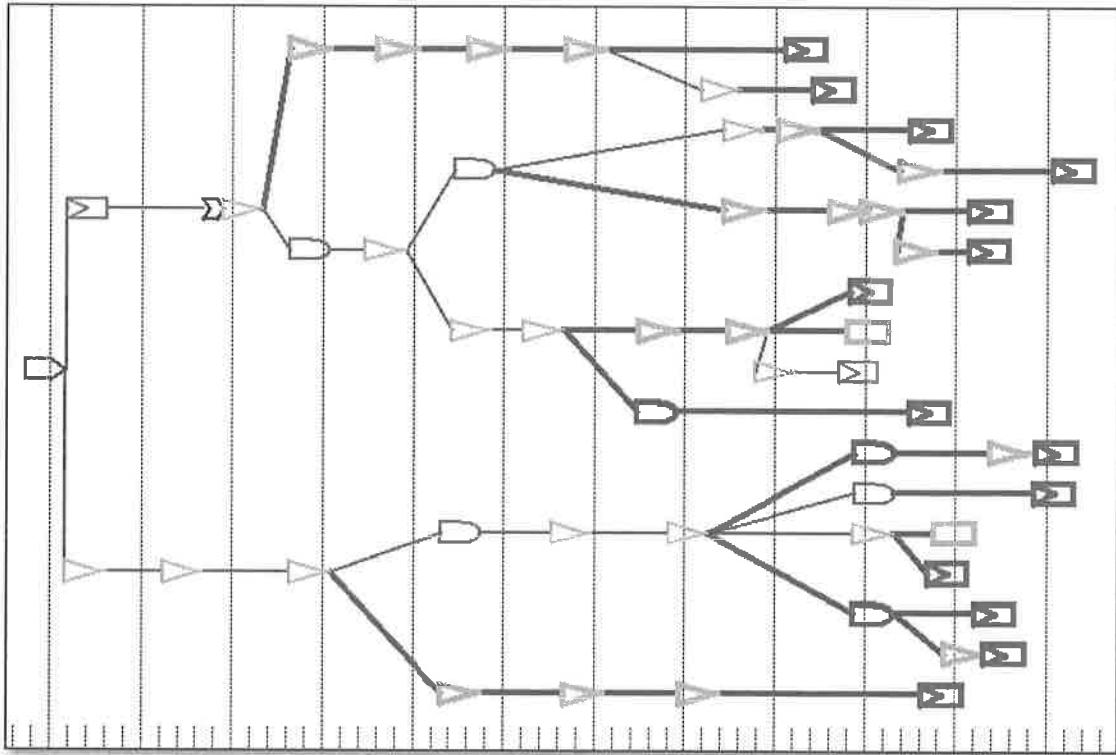
1. Have a look at the synthesized clock tree using the *clock abstract graph* GUI.
In the GUI select **Window→Clock Tree Analysis Window**.
2. In the new *CTSWindow*, check the little box in the top right corner next to '**Clocks associated with corner**'. This allows us to analyze clock latencies, which vary by corner. Make sure the **ss_125c** corner is selected.
3. In the main section of the window, where all the clocks are listed, go to the **func** scenario section, right click on the **sys_2x_CLK**, then select **Clock Tree Latency Graph of selected Corner**.

This screenshot shows the latency graph for **SYS_2x_CLK** for the classic CTS flow (the CCD screenshot is shown on the next page):



Lab 7

This screenshot shows the latency graph for `SYS_2x_CLK` for the CCD flow:



As expected, the latency distribution to the register endpoints will be larger for CCD than for classic CTS.

4. Close the *CTSWindow*.
5. Examine the clock tree routing topology, by selecting the **Clock Tree** visual mode.

In the panel that opens on the right, click on the little gear symbol in the upper-right corner, to show the panel settings, then make sure that “Auto Apply” is selected.

From the list of clocks, select an individual clock to visualize its routing topology in the layout window.



You can also select how many, and which levels of the routing topology to show.

Note: Level 0 is the net that connects to the clock root or source.
 Level 1 is the net that is driven by the first driver, etc.

6. Close the *Clock Tree* visual mode panel.
7. Examine the timing from one of the clocks constrained by v_PCI_CLK:

```
report_timing -from [get_clocks v_PCI_CLK]
```

Question 14. Is the network latency on the input v_PCI_CLK propagated?

.....

You have successfully completed the Running CTS lab.



Answers / Solutions

Question 1. From a timing stand-point – is the design ready for CTS? What about the hold violations?

There should be no *setup* violations. There are *hold* violations, which will be addressed during CTS.

Question 2. How many *master* clocks are defined?

There are 3 *master* clocks defined: Master clocks have a *clock source*, and are not *generated* (“G”).

PCI_CLK, SDRAM_CLK and SYS_2x_CLK

Question 3. How many *generated* clocks are defined?

3 clocks are *generated* clocks:

SD_DDR_CLK, SD_DDR_CLKn, and SYS_CLK.

Look at the top section of the report:

The **Attrs** column shows the letter G (generated).

The section below the list of all clocks contains details of the *generated* clocks.

Question 4. What type of clock are the remaining clocks?

The remaining clocks are *virtual* clocks. They are:

v_SDRAM_CLK and v_PCI_CLK

Virtual clocks have an empty field in the **Sources** column.

Question 5. What is the **Source** of the SD_DDR_CLK clock?

From the **Sources** column in the top section of the report: **sd_ck**, which is a generated clock of **SDRAM_CLK**.

Question 6. What is smallest/largest Setup Uncertainty?

What is smallest/largest Hold Uncertainty?

From **report_clocks -skew**:

Setup Uncertainty:

0.10 ns (*ff* scenarios), 0.30 ns (*ss* scenarios)

Hold Uncertainty:

0.05 ns (*ff* scenarios), 0.10 ns(*ss* scenarios)

Question 7.

Which clock groups are *mutually exclusive* or *asynchronous*?

The report lists three *asynchronous* clock groups:

SYS_2x_CLK and **SYS_CLK**

PCI_CLK and **v_PCI_CLK**

SDRAM_CLK, **v_SDRAM_CLK** and **SD_DDR_CLK***

All clocks within a group are *synchronous* to one another, but clocks across groups are *asynchronous* to one another. Clock groups can be *logically_exclusive*, *physically_exclusive*, or *asynchronous*: Paths between exclusive or asynchronous clocks are not considered during timing analysis (similar to `set_false_path`). In addition, *logically_exclusive*, *physically_exclusive*, and *asynchronous* relationships affect how crosstalk analysis is performed between the clocks.

Question 8.

What is the big difference between the two modes?

(Hint: Look at the clock names)

The `test` mode has an additional clock: **ate_clk**.

Question 9.

How many sinks does **SD_DDR_CLK** have?

Zero sinks.

Question 10.

Why does **SD_DDR_CLK** have zero *sinks*?

(Hint: Look at the port direction `Dir`)

`SD_DDR_CLK` is a generated clock defined on the port `sd_CK` (as shown by `report_clock`). From `report_port [get_ports sd_CK]` you will see that this is an output port, and therefore has no sinks.

Question 11.

Are the scenarios configured properly for hold fixing?

You will find that there are three `*ff*` scenarios, of which one is not configured for hold fixing: `test.ff_125c`. Since this is a fast scenario, make sure it is enabled for hold fixing.

Question 12. Why are the skews reported by `report_clock_qor` and `report_clock_timing` different?

The `report_clock_qor` command reports global skew, by default, which is the maximum skew across all sinks in 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_qor` does not use timing derates, even though they might be defined in your corners. You can also generate a local skew report using `report_clock_qor -type local_skew`.

The `report_clock_timing` report only calculates local clock skew: The reported maximum skew is between two flip flops 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 timing derates, but for setup timing it applies late derates to the launch clock path and early derates to the capture clock path. The latter report is more accurate in determining the real worst-case clock skew.

Question 13.

Are there any setup or hold violations left?

There should only be a few small setup and hold violations.

Question 14.

Is the network latency on the input `v_PCI_CLK propagated`?

No. A latency number is listed under “`clock network delay (ideal)`”. This is the auto-updated *ideal* latency value that *ICC II* has computed for `PCI_CLK`, and applied to `v_PCI_CLK` (since we configured this in Task 3).

8

Setting up CTS

Learning Objectives

The purpose of this lab is for you become familiar with the setup steps for clock tree balancing, NDRs as well as timing/DRCs.

After CTS setup, you will run the `build_clock / route_clock` clock tree synthesis stages to confirm the results.

After completing this lab, you should be able to:

- Set up clock tree balancing
- Create and apply Non-Default routing rules
- Apply clock-related timing and DRC constraints



Lab Duration:
45 minutes

Lab 8

Instructions

Answers / Solutions

You are encouraged to refer to the back of the lab to verify your answers, and to look for hints.

Task 1. Load the Design and Analyze the Clocks

1. Change to the work directory for the CTS lab, then load the starting design:

```
UNIX% cd lab8_cts  
UNIX% icc2_shell -gui -f load.tcl
```

The script will open a block that is ready for the upcoming tasks.

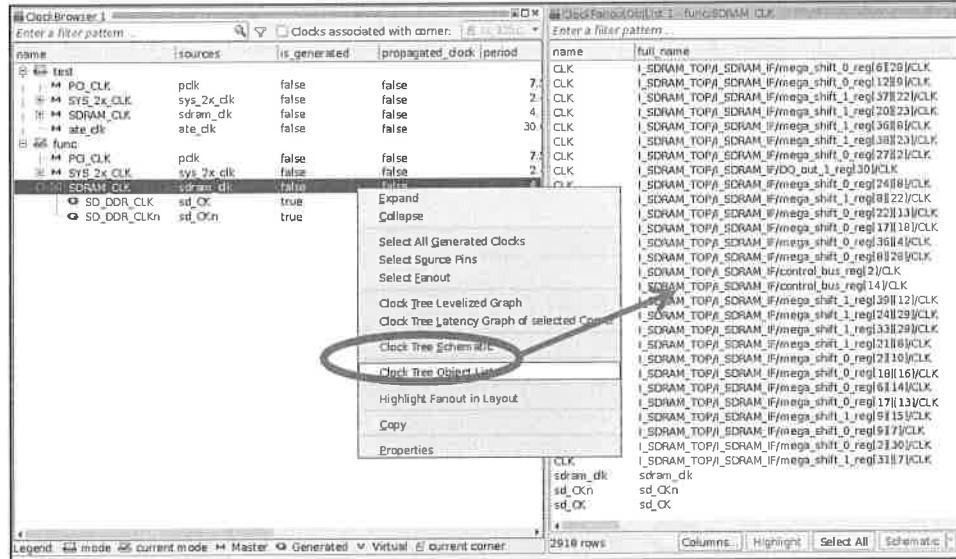
2. Open the **run.tcl** file in the *ICC II script editor*, as in previous labs.

This file contains all the commands that you will be executing in this lab. Select the commands from this file and use **Run Selection**, instead of typing them yourself, to save time and avoid typing errors.

3. Select the menu **Window→Clock Tree Analysis Window** and answer **Yes** when asked “... Do you want to continue?”.

4. Expand the **SDRAM_CLK** entry (click on the “+” in front of it) and you will see the two **SD_DDR_CLK*** clocks. Notice that the **Is Generated** column is be set to true for these clocks, and their **Sources** are **sd_ck***. Also, you will see “M” and “G” symbols in front of the clocks, which identifies them as *Master* or *Generated* clocks, respectively. In summary: The **SD_DDR_CLK** and **SD_DDR_CLKn** clocks are *generated* from the *master* clock **SDRAM_CLK**. The **SDRAM_CLK** clock is applied to its *source* port **sdram_clk**. The generated clocks are applied to their *source* ports **sd_ck** and **sd_ckn**, respectively.

5. Perform closer analysis of **SDRAM_CLK** in **func** mode by right-clicking on it, then selecting “**Clock Tree Object List**”. In the “Find CTS Object” dialog that appears, select **OK**.



You should see a new window as shown above.

In this new window, you will see all *sink* pins of this clock. If you scroll all the way to the bottom, you will also see the output ports `sd_CK` and `sd_CKn`.

If you scroll to the right, you will see many more attributes associated with the pins/ports. To better display the information that is important to you, you can move the position of the columns to the left/right, and you can sort the order by the values in any column.

Close the CTS window.

6. To report what type of *balance points* exist on the clock tree endpoints (implicit or explicit *ignore* or *sink* pins), generate a *clock structure* report:

```
v report_clock_qor -type structure
```

In the view window, type **Ctrl-F** (or click on **Search...**) then enter the search string “**sd_CK**” and press enter. You should see the line beginning with `sd_CK [out Port]`, and at the end of the line you will see a balance point *exception* (something other than an implicit `SINK PIN`).

Question 1. What balance point exception is set on `sd_CK`, and why?

.....
.....

Do NOT close the *view* window yet.

Lab 8

Task 2. Clock Tree Balancing

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 *sink pins*, which, by default, are *clock pins* of sequential cells or macros. If there are additional pins that need to be balanced along with these clock pins, *ICC II* needs to be explicitly told about them prior to *CTS*.

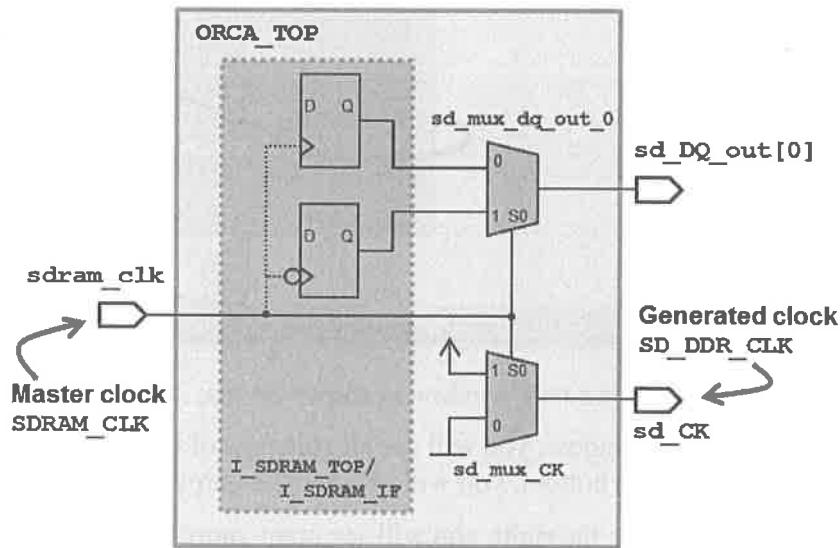


Figure 1. SDRAM interface

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 ports of the `ORCA_TOP` block. 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_out` and `sd_CK` be optimized. By default, select (`S0`) pins are marked as *implicit ignore pins*. To have *CTS* balance the skew you need to redefine these select pins as *sink pins*.

1. In the *view* window that should still be open (see the last step of the previous *Task*), notice that just above and below the `sd_CK` line, the MUX *select* pins described in *Figure 1*, `I_SDRAM_TOP/I_SDRAM_IF/sd_mux_ck*/S0`, are, in fact, also *implicit ignore pins*.
2. Click on **Dismiss Search** and then **Exit** the *view* window.

As a reminder, select and run the commands from the `run.tcl` script using the built-in script editor.

3. Apply balancing constraints for the S0 pins in all modes:

```
foreach_in_collection mode [all_modes] {
    current_mode $mode
    set_clock_balance_points \
        -consider_for_balancing true \
        -balance_points [get_pins \
            "I_SDRAM_TOP/I_SDRAM_IF/sd_mux_*/S0"]
}
```

4. Generate the following report to verify the user-defined balance points:

```
report_clock_balance_points
```

You should see the pins listed below the following lines:

Clock Independent:

Balance Points:

The reason the balance points are “Clock Independent” is because you did not specify a clock to go with the exception. If the exception is intended to be balanced with regard to a specific clock, and there are multiple clocks reaching this point, then a clock should be specified. This is not the case here.

5. Have another look at a *clock structure* report, and search for `sd_mux`:

```
v report_clock_qor -type structure
```

6. In the *view* window, search for `sd_CK`.

Question 2. How are the S0 (select) pins of the MUXes labeled now?

Question 3. How is the `sd_CK` port labeled now? What does this mean?

7. In the *view* window, click on **Dismiss Search** and the **Exit** the *view* window.

Lab 8

8. Instruct CTS to not change the SDRAM muxes: They have been chosen to achieve the best timing, and should be left as they are.

```
set_dont_touch \
[get_cells "I_SDRAM_TOP/I_SDRAM_IF/sd_mux_*"]
```

9. Verify your settings with the `report_dont_touch` command:

```
report_dont_touch I_SDRAM_TOP/I_SDRAM_IF/sd_mux_*
```

10. Return to **Window→Clock Tree Analysis Window**. Expand the `SYS_2x_CLK` (click on the “+” in front of it). You should see the `SYS_CLK` clock.

Question 4. What is the source of this *generated* clock?

.....

11. Instruct CTS to not change the register that is used as the clock divider:

```
set_dont_touch \
[get_cells "I_CLOCKING/sys_clk_in_reg"]
```

12. Verify with the reporting command used in a previous step.
13. Set a **skew target** of **0.05ns** for all the slow (`ss`) corners, and **0.02ns** for the fast (`ff`) corners.
14. Generate the following report and confirm:

```
report_clock_tree_options
```

15. When performing CTS, it is generally desirable to use specific cells for synthesis, instead of letting *ICC II* choose any cell from the library, for example: Cells which help to reduce skew (identical rise/fall ramp times); Cells which help to better balance between power consumption and speed/drive-strength, size, etc.

CTS-specific cells are defined using:

```
set_lib_cell_purpose -include cts
```

First, automatically identify the gates and ICGs that are already on the clock network, and their *logical equivalents* (leq's):

```
derive_clock_cell_references -output cts_leq_set.tcl
```

Note that the above will only work if all library cells already have the cts purpose.

16. Have a look at the file that was created - `cts_leq_set.tcl`.

As you can see, all cells that are on the clock network currently have been identified, along with their *LEQ*'s. You could copy and paste the commands to a new file, uncomment the appropriate lines, and source it later, however, *you do NOT have to do this - we have already done this for you*.

17. Next, choose the buffers and/or inverters you want to use for the clock tree – this is done using the following lines:

```
set CTS_CELLS [get_lib_cells      \
    "*/NBUFF*LVT */NBUFF*RVT * \
     /INVX*_LVT */INVX*_RVT /*DFF*"]
set_dont_touch $CTS_CELLS false
set_lib_cell_purpose -exclude cts [get_lib_cells]
set_lib_cell_purpose -include cts $CTS_CELLS
```

Select/run these lines from `run.tcl`.

18. Instead of sourcing an edited copy of the `cts_leq_set.tcl` file that you created earlier, you can source our version:

```
source scripts/cts_include_refs.tcl
```

19. Generate a report to ensure that the correct lib-cell purpose was indeed set, and `dont_touch` was removed, on the key CTS cells:

```
v report_lib_cells -objects [get_lib_cells] \
    -columns {name:20 valid_purposes dont_touch}
```

In the view window, search for the string “`cts`”.

You could also use the workshop-provided alias `_full_lib_report`.

Lab 8

Task 3. Define CTS Non-Default Routing Rules

In this task you will specify CTS non-default routing rules, as well as clock cell spacing rules.

1. Open the file `scripts/ndr.tcl` in an editor.
2. Review the file and answer the following questions:

Question 5. Which net segment(s) of the clock tree do the two clock routing rules apply to?

.....
.....
.....

Question 6. What are some key differences between the rules?

.....
.....
.....
.....
.....

3. Apply the clock NDRs:

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

4. First, verify that the routing rules that were created:

```
v report_routing_rules -verbose
```

You should see the metal layer details for each of the two rules that were created. In addition, you should see a section for the vias.

5. Now verify where the rules were applied:

```
report_clock_routing_rules
```

The report shows which net segments (`net_type`) the rules apply to (`sink overrides all`), and the min/max layer constraints for each clock segment.

Task 4. Timing and DRC Constraints

1. Generate a port report for the master clock sources:

```
report_ports -verbose [get_ports *clk]
```

Verify that the master clock sources are input ports, and that they are all constrained by either a Driving Cell or input Transition.

Question 7. Are all clock ports constrained by either a Driving Cell or input Transition?

.....

.....

Question 8. Why is it important for clock input ports to be constrained by set_driving_cell or set_input_transition?

.....

.....

2. Fix the problem found above by adding a driving cell to the ate_clk port. Driving cells need to be added in all scenarios, because this constraint is *scenario-specific*.

Specify NBUFFX16_RVT as the driving cell for the port ate_clk, then report the clock ports again:

```
foreach_in_collection scen [all_scenarios] {
    current_scenario $scen
    set_driving_cell \
        -lib_cell NBUFFX16_RVT [get_ports ate_clk]
}
report_ports -verbose [get_ports *clk]
```

Lab 8

3. Take another look at the clock *uncertainty* numbers using `report_clocks -skew`. Next, we will change the clock *uncertainty* numbers to account for post-CTS *propagated* clock timing. This is accomplished by reducing the uncertainty by the value that was intended to model the *skew*. The *uncertainty* should still model the effects of clock *jitter* or additional timing *margin*.

Apply the following commands:

```
foreach_in_collection scen [all_scenarios] {  
    current_scenario $scen  
    set_clock_uncertainty 0.1 -setup [all_clocks]  
    set_clock_uncertainty 0.05 -hold [all_clocks]  
}
```

4. Apply a ***max transition*** constraint of **0.15ns** on all clocks, in all corners of the func mode.
5. Enable the **removal of *clock reconvergence pessimism*** to eliminate the timing pessimism of *OCV timing derating* on shared launch/capture clock tree branches.
6. Generate the following report to confirm the *max transition* setting:

```
v report_clock_settings
```

Note: To see the correct *max transition* information, you have to scroll down past the `##Global` section, and search for the “Mode = func” section which lists all the individual clocks. The report first lists Global settings for all modes/corners, which were not set in our case. Instead, we applied clock-specific settings (to all clocks) by using “`-clock_path [get_clocks]`”, in the current `func` mode.

Task 5. Perform CTS and Analyze the Results

1. Build and route the clock trees:

```
clock_opt -to route_clock
```

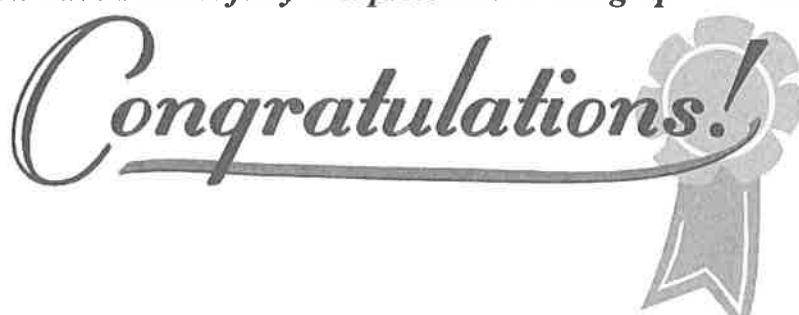
2. In the GUI, turn off the visibility of power and ground nets, and zoom in to have a closer look at the clock routes. If you hover the mouse cursor over a net, in the *query* window that appears, you will be able see the NDRs *routing rule* that has been applied.
3. Report the *skew* between all the *sd_mux** pins, which were defined as *sink* pins in an earlier *Task*. An easy way to do this is:

```
report_clock_qor \
    -to I_SDRAM_TOP/I_SDRAM_IF/sd_mux_*/S0 \
    -corners ss_125c
```

You should find that the *global skew* is pretty small.

4. Have a look at the *clock tree latency graph* for *SDRAM_CLK*:
 - a. In the GUI: **Window → Clock Tree Analysis Window**
 - b. Check the little box in the top right corner next to '**Clocks associated with corner**'. This allows us to analyze the latency, since latency calculation is done on a corner-by-corner basis (without selecting a corner, the x-axis of the latency graph displays "levels of logic" instead).
 - c. Right click on *SDRAM_CLK* under the *func* scenario, then select **Clock Tree Latency Graph of selected Corner**.
5. Time permitting, perform any additional analysis which is of interest to you.

You have successfully completed the Setting up CTS lab.



Answers / Solutions

Question 1. What balance point exception is set on `sd_CK`, and why?

It is an [**IMPLICIT IGNORE PIN**]. Clock tree *sink pin* endpoints are register or macro clock pins, by default. This is an output port, and therefore ignored for balancing, by default.

Question 2. How are the `S0` (select) pins of the MUXes labeled now?

[**BALANCE PIN**], which represents explicit, or user-defined *sink pin* exceptions.

Question 3. How is the `sd_CK` port labeled now? What does this mean?

The `sd_CK` port is now described as [**BEYOND EXCEPTION**], since the port is in the fanout of (beyond) the **BALANCE PIN** exception on the mux. CTS non-default design rules, as well as general max transition and capacitance design rules will be applied on the “beyond exception” sections of the clock network.

Question 4. What is the source of this *generated* clock?

The source is `I_CLOCKING/sys_clk_in_reg/Q`. This is a divide-by-2 register, used to divide `SYS_2x_CLK` and generate `SYS_CLK`.

Question 5. Which net segment(s) of the clock tree do the two clock routing rules apply to?

The bottom routing rule `$CTS_LEAF_NDR_RULE_NAME` (`cts_w1_s2`), is applied to the sink segments of the clock net (`-net_type sink`).

The top rule `$CTS_NDR_RULE_NAME` (`cts_w2_s2_vlg`), is applied to the remaining root and internal segments of the clock net, since `-net_type` was not specified.

Question 6. What are some key differences between the rules?

The \$CTS_LEAF_NDR_RULE_NAME rule for the sink nets specifies only non-default spacings (uses default widths), while the \$CTS_NDR_RULE_NAME rule has both non-default spacings and widths, as well as via (cuts) rules

The sink nets will be routed on M1-M5, while the root and internal clock tree nets will be routed primarily on M4-M5.

For the sink rule, M1 non-default spacing is not specified: This is to avoid potential DRC violations when connecting to the standard cell pins.

For the root and internal clock tree nets, NDRs were included for the lower non-primary CTS routing layers M1-M3: This is generally recommended, since these nets will have to eventually be routed on these lower layers to connect to the clock pins.

Question 7. Are all clock ports constrained by either a Driving Cell or input Transition?

No. You will find that the clock port ate_clk appears in most report sections, but not in the Driving Cell or Transition sections (Note: The Transition section appears only if at least one input has an input transition applied, which is not the case here). This means that there is no *driving cell* or *input transition* constraint for this clock port. The reason you did not see this port before is because it is only defined as a clock in test mode.

Question 8. Why is it important for clock input ports to be constrained by set_driving_cell or set_input_transition?

During CTS, clock insertion delays are calculated to optimize for skew and latency. Either one of the above two constraints allows CTS to more accurately calculate the delay of the first gate connected to the clock input port, for more accurate clock insertion delay calculation (a cell's delay is a function of its input transition time). Input port transition time is 0ns, by default, which can cause less accurate results.

This page was intentionally left blank.

11

Routing and Post-Route Optimization, Signoff

Learning Objectives

After completing this lab, you should be able to:

- Perform routeability checks on a placed design with clock trees
- Apply routing options
- Route secondary PG nets
- Control via optimization
- Perform route and post-route optimization
- Analyze the design for timing with SI enabled, and perform incremental power and crosstalk optimizations
- Optional: Perform sign-off DRC checking and fixing, standard cell filler insertion, and sign-off metal fill insertion.



Lab Duration:
45 minutes

Instructions

Answers / Solutions

You are encouraged to refer to the back of the lab to verify your answers.

Task 1. Load and Check the Post-CTS Design

1. Change to the work directory for the Routing lab, then load the post-CTS design:

```
UNIX% cd lab9_11_route_signoff  
UNIX% icc2_shell -gui -f load.tcl
```

The script will make a copy of the clock_opt block and open it.

2. Generate a timing QoR summary.

```
report_qor -summary
```

Question 1. Is timing acceptable for routing?

.....
.....
.....

Task 2. Route the Design

To support a more immersive and interesting lab experience, *there are no step-by-step instructions for this lab.*

Instead, you are asked to open the file `run.tcl` in an editor (or using the *ICC II script editor*), and exercise the commands line by line. We are specifically asking you not to just source the entire file, as this would defeat the purpose, which is to understand how all the options and commands play together. If there is an option that does not make sense, have a look at its man page.

The following sections provide additional information and comments. The sections are ordered in such a way that you can refer to them as you go through the script.

If you like, you can diverge from the commands in `run.tcl`, or you could try different efforts, different settings etc. Note, though, that the runtimes might vary. All in all, the runtimes are very quick, so you are encouraged to experiment.

The following section is designed to be a guide through the lab. It contains information on the items that have to be configured and run, and some additional questions.

If you need help, talk to your instructor.

Pre-routing Checks

Before you route the design it is best to ensure that there are no issues that will prevent the router from doing its job.

Question 2. Is the design ready for routing?

.....
.....
.....

Antenna

Antenna definitions are commonly supplied in a separate *TCL* file, specific to the technology, using the following commands (these are the same commands used in *IC Compiler*):

- `define_antenna_rule`
- `define_antenna_layer_rule`
- `define_antenna_area_rule`
- `define_antenna_accumulation_mode`
- `define_antenna_layer_ratio_scale`

In addition, there are application options that control how antenna violations are handled. Use the `report_app_options` command shown in `run.tcl`.

Lab 11

Crosstalk Prevention

Crosstalk prevention tries to ensure that timing-critical nets are not routed in parallel over long distances. Prevention can occur in the *global routing* and the *track assign* stages. The current recommendation is to enable prevention during the track assign stage only. In order for crosstalk prevention to occur, crosstalk (or signal integrity) analysis must also be enabled. In order to make post-route analysis and optimization more interesting (showing SI violations that need to be fixed during `route_opt`), you can choose to do that “artificially” by not enabling SI analysis during routing.

Secondary PG Routing

Secondary PG pins are power pins on special cells like *level shifters* or *isolation cells*. In addition to the regular power supply provided through the standard cell rails, these secondary power pins need to be routed to using the standard router. There are a few routing parameters used to configure the routing behavior. In addition, it is common to define non-default rules for these power connections.

After performing the secondary PG routing as shown in the script, identify the low-to-high level shifters (in the `PD_RISC_CORE` voltage area) and examine their power routing, to answer the following question:

Question 3. What is the name of the secondary PG pins, and where do they connect to?

.....
.....
.....

Routing, DRC Analysis

After completing secondary PG routing, as well as routing “critical” signals, you perform auto-route. At this stage this will route all signal nets that have not been routed previously. Any signals that have been routed already (clocks, secondary PG) will not be touched again if they are DRC clean. Auto-routing runs *global routing*, *track assignment* and *detail routing*.

Question 4. How many detail route iterations are run by default?
(Hint: Review the man page for `route_auto`.)

.....

Question 5. How do you change the default, and how do you force the router to run through all iterations even though the router might not see any improvements
 (Hint: `report_app_options *iterat*`)?

.....

Examining Routing Design Rule Violations

Use the *error browser* to visualize any routing violations in the GUI that may have been left over. On the top toolbar, select the button for the *error browser*.



In the popup, select `zroute.err` and click on **Open Selected**. In the new window, you can select the errors from the list, which causes the layout view to zoom to that violation.

Close the Error Browser.

Signal Integrity

Signal integrity analysis should be turned on before routing. This will instruct the extraction engine to extract cross-coupling capacitances, and instruct the router to perform timing analysis with *delta delays* using these coupling capacitances. This is important when performing timing-driven routing.

For this lab, if you left SI analysis off before auto-route, turn it on afterwards to see the SI effects, and to allow SI-related violations to be fixed during `route_opt`.

For better correlation with PrimeTime, you should also enable timing window analysis, as shown in the script.

Post-Route Optimization

Post-route optimization is performed using `route_opt`, which performs timing, logical drc, area and (optionally) CCD and power optimization.

There are application options you have to set in order to enable CCD and power optimization.

For best correlation with PrimeTime, you should enable PT delay calculation. In this lab, don't perform StarRC InDesign extraction in `route_opt`. You will be using StarRC later when performing ECO Fusion.

Lab 11

ECO Fusion

After completing `route_opt`, it's important to analyze signoff timing in PrimeTime using StarRC parasitic extraction. Any violations uncovered by PT can be fixed using PT's physical ECO.

Using Fusion, the entire ECO process can be controlled from within ICC II.

Review the `run.tcl` file for the necessary commands, and perform one round of ECO fixing. For Fusion, you will require ICC II, StarRC and PrimeTime SI.

Note that after the ECOs have been implemented, you have to analyze timing using the command `check_pt_qor`. You should not run ICC II's native timing reporting commands (`report_qor`, `report_timing`, ...).

Std Cell Fillers, ICV DRC, Metal Fill

After all post-route optimizations are complete, and any necessary ECOs have been performed, perform standard cell filler insertion as shown in the `run.tcl` script.

If you want to perform signoff DRC checking, have a look at the next steps. You will find commands for performing DRC checking and also ICV DRC auto-repair.

Please use the `select_rules` filter as shown, otherwise you will see many violations which are due to a mismatch between our techfile and the ICV DRC runset.

You can review the errors generated by ICV using the error browser. If the error browser is already open, use **File→Open...** and select `signoff_check_drc.err`. Otherwise select the error file in the error-browser popup (the *Checker* column will say "IC Validator").

As the final step, perform metal filling using *IC Validator*. Have a look at the very end of the `run.tcl` script.

Examine the metal fill using the GUI, as described in the lecture.

You have successfully completed the Routing lab.



Answers / Solutions

Question 1. Is timing acceptable for routing?

You should see that there are only a few remaining small timing violations. There are also a few smaller *max transition* as well as a few *max capacitance* violations (you will see them with `report_constraints -all`)

Question 2. Is the design ready for routing?

Using the `check_design` command shows a few issues. Examine the EMS issues in the message window by using **Window→Message Browser Window**, then use **File→Open Message Database...** to open the file `check_design.ems`.

For this lab, you can ignore the EMS messages.

For the remaining non-EMS messages, you have to review the file `check_design*.log`.

ZRT-022 warns about a missing default contact for layer CO, which is entirely fine for this technology. ZRT-044 lists a few cells that do not have valid via regions, which is something that should be addressed in the corresponding physical library during library preparation. ZRT-585 complains about cell-internal pins, that will not be relevant to the router. Finally, please ignore ZRT-511.

Question 3. What is the name of the secondary PG pins, and where do they connect to?

The PG pins for the low voltage are called **VDDL**. The router connects them to the nearest **VDD** vertical straps, or horizontal rails, which are located just outside the voltage area boundary. You can verify the voltages using `report_power_domains`.

Question 4. How many detail route iterations are run by default?

There, the default is stated as **40**.

Question 5. How do you change the default, and how do you force the router to run through all iterations even though the router might not see any improvements?

You change the default using
`route_auto -max_detail_route_iterations <#>`
To force `route_auto` to actually run through all iterations you need to change an application option from its default (`false`):
`route.detail.force_max_number_iterations → true`