MAVEN SILICON

Centre of Excellence in VLSI

Physical Design using Fusion Compiler Lab Manual

www.maven-silicon.com

Maven Silicon Confidential

Table of Contents

Lab 01 - Using Fusion Compiler GUI

Objective : *Explore and familiarize yourself with Fusion Compiler GUI.*

Working Directory : /home/<user_name>/PD_Labs/FC/lab01

Project Name : ORCA_TOP.lib

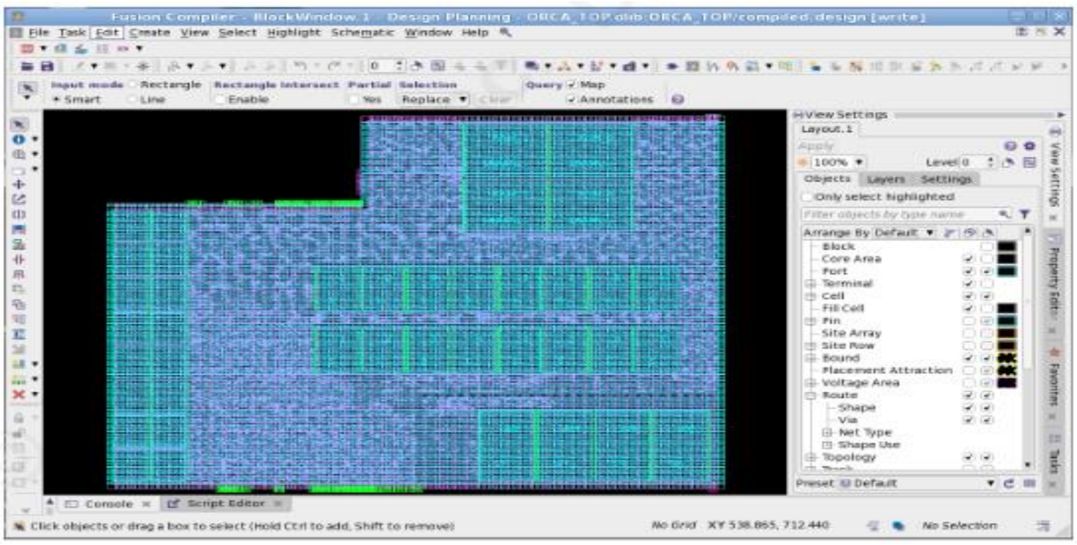
Learning outcomes:

- ✓ Invoke and exit the Fusion Compiler GUI
- ✓ Navigate the *layout view*
- ✓ Control object visibility and selectability
- ✓ Select and query layout objects
- ✓ Control object view level
- ✓ Rearrange panels in the GUI
- ✓ Use the recent menu and favourites
- ✓ Perform timing analysis
- ✓ Use *Command Search*, as well as *help* and *man* to get help and additional information about commands and options.

Task 1: Launch Fusion Compiler and open a Block

Instructions: Follow the below instructions and answer the given questions based on your observations after the execution of the relevant instructions.

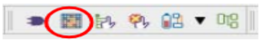
- ✓ Invoke the Fusion Compiler tool with ***fc_shell***
- ✓ Start the GUI ***start_gui***
- ✓ Block windows open and many GUI commands can be invoked by selecting icons, pull-down menus, or by using hotkey shortcuts.
- ✓ Load the project using the pull-down menu: ***File*** ➔ ***Open Block,***
- ✓ Click on the yellow symbol in the top-right corner and choose a design library named "ORCA_TOP.dlib".
    - **Note:** The design libraries are identified by this symbol 🏠
- ✓ Now that a library has been chosen, select "ORCA_TOP/compiled" from the blocks listed in the **open Block** dialog and click **OK**. (see the layout view of ORCA_TOP in the GUI).



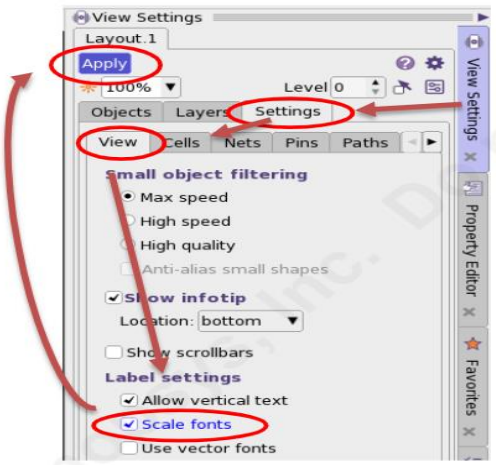
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

No Grid XY 538.865, 712.440 No Selection

- ✓ The Layout can always reopen by clicking on the **Layout View** button from the top bar or by selecting the menu ***View*** ➔ ***layout View***



- ✓ To enable the size of the text in the Layout View to scale when zooming in, select the **settings** tab under **View Settings**, then the **View** tab. There, under **Label settings**, check **Scale fonts** and Apply.
- ✓ Turn on "**auto-apply**" as follows, click on the **show options** gear icon on the top-right of the view settings tab, then select **Auto apply**

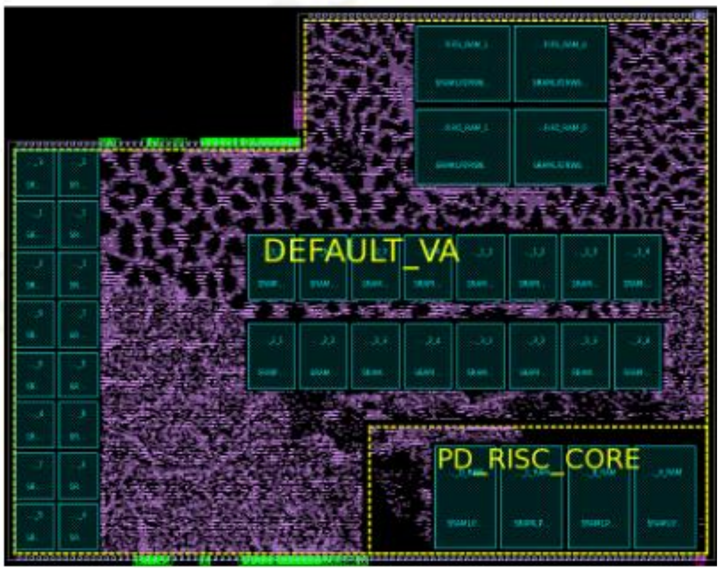


Task 2: Layout View Settings

- ✓ Perform the following steps to turn off the visibility of the power mesh, and to turn on the voltage area outlines-
  - In the View Settings, select the **objects** tab
  - To the right of **Route** object, **uncheck** the left "Toggle object visibility" box. The power mesh should no longer be visible.
  - **Check** the **Voltage Area** visibility box. You should now see the dashed-yellow outlines of the DEFAULT_VA and PD_RISC_COREvoltage areas.
  - Change the color of voltage area outlines and labels to make them more clearly visible: click on the dashed-purple icon on the right of the voltage area and chose **Yellow.**

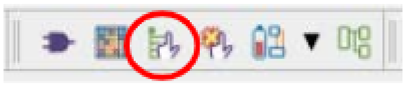
The Layout should look as shown below.

- Observe the I/O logical ports and P/G physical terminals along the boundary edges.

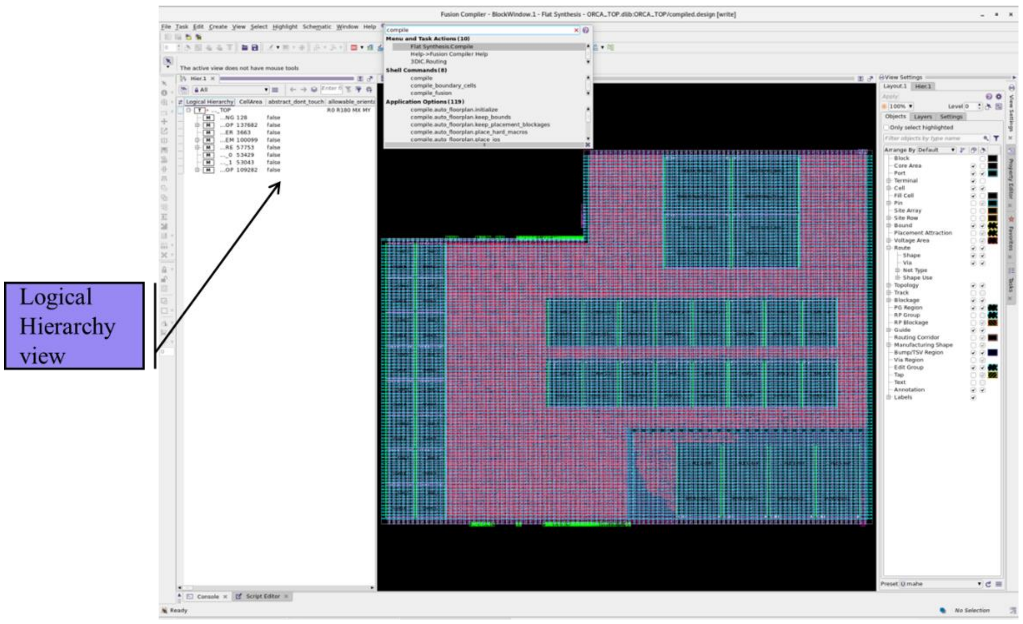


Task 3: Explore the Logical Hierarchy

- ✓ Open the hierarchy browser by clicking on the button shown from the top bar or by selecting the menu **View → Hierarchy Browser**.

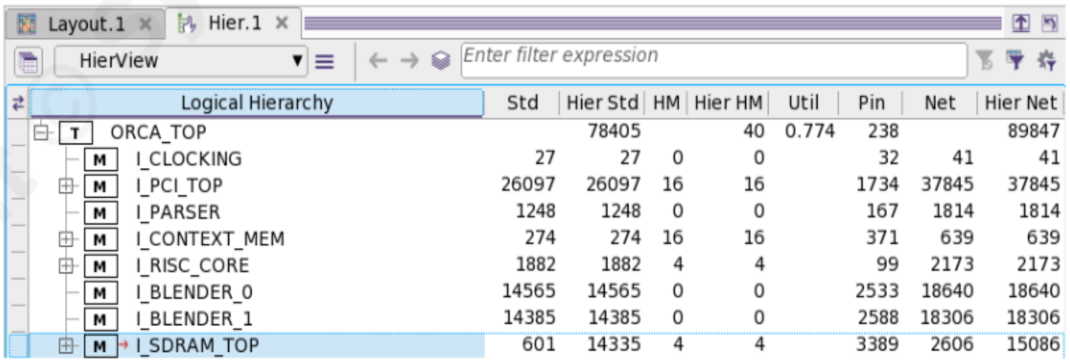


The block window looks like as shown below



The Hierarchy Browser lists cell (instance) names of sub-design modules (**M**) and hard macros (**HM**), along with their statistics.

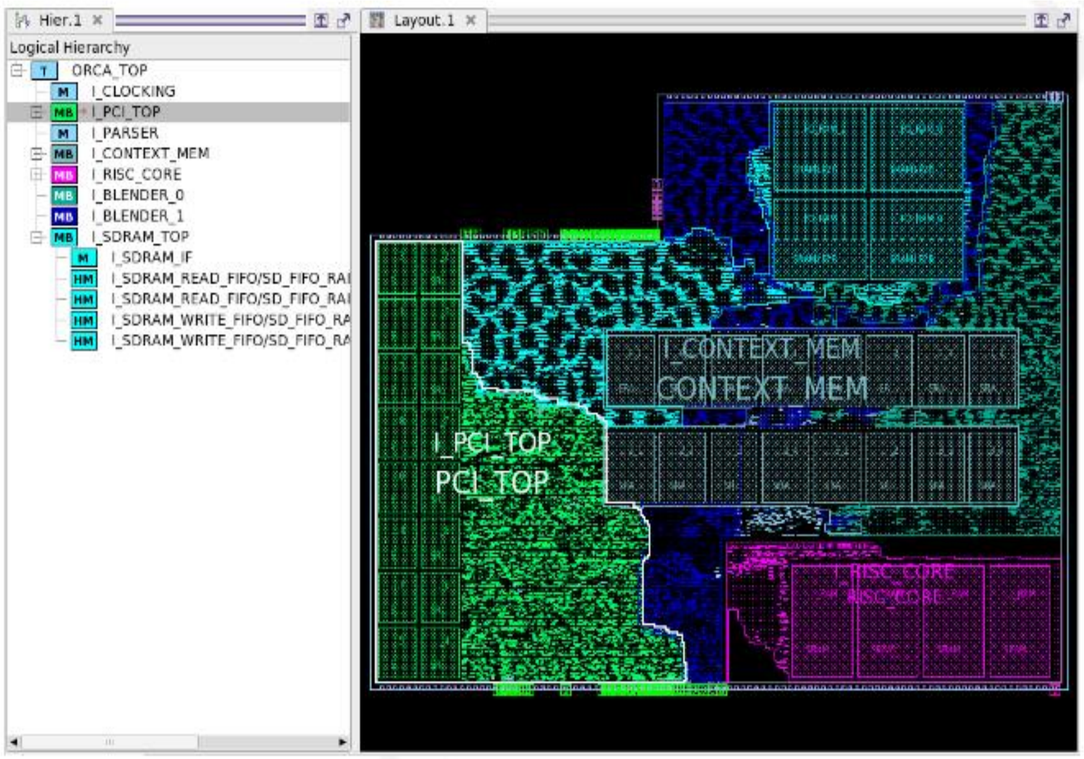
- ✓ Expand the width of the hierarchy tab to see the columns displaying the statistics, or **double-click** on the title bar (to the right of *Hier.1*) to maximize the view.
- ✓ The following statistics are displayed; Utilization percentage; pin count; hard macro, standard cell, and net count (at the current level only and hierarchically).


Layout.1
Hier.1
HierView
Enter filter expression

Logical Hierarchy	Std	Hier Std	HM	Hier HM	Util	Pin	Net	Hier Net
T ORCA_TOP		78405		40	0.774	238		89847
M I_CLOCKING	27	27	0	0		32	41	41
M I_PCI_TOP	26097	26097	16	16		1734	37845	37845
M I_PARSER	1248	1248	0	0		167	1814	1814
M I_CONTEXT_MEM	274	274	16	16		371	639	639
M I_RISC_CORE	1882	1882	4	4		99	2173	2173
M I_BLENDER_0	14565	14565	0	0		2533	18640	18640
M I_BLENDER_1	14385	14385	0	0		2588	18306	18306
M I_SDRAM_TOP	601	14335	4	4		3389	2606	15086

- ✓ Click on the "+" sign to the left of the I_SDRAM_TOP module to expand its hierarchy.
- ✓ Select one of the hard macros (HM). Notice that the selected hard macro is highlighted with a white "X" in the layout view. You can select and highlight multiple hard macros by using the [Ctrl] or [Shift] keys.
- ✓ Unselect the hard macro(s) by clicking the cursor in any black area around the layout or pressing the hotkey [Ctrl-D]
- ✓ Right-click on any module (M), and from the context menu select Color by Hierarchy → **Initialize Color by Hierarchy.**

The hierarchies/modules were highlighted in the layout using a different color as shown below



- **Note:** Fusion compiler will draw Hierarchical module boundaries (**MB**) around the various modules in the design.
- Smaller modules are not highlighted in the GUI to reduce clutter, for example, the module I_PARSER remains as **M**, and if you select I_PARSER nothing is highlighted in the layout view.

✓ Right-click on the `I_PARSER`module, and from the context menu select **Color by Hierarchy** ➔ **Create Selected**. The `I_PARSER`module is now shown as MB, and the corresponding standard cells can be seen in a new color in the layout view.

✓ To disable all coloring, right-click a module then select **Color by Hierarchy** ➔ **Remove All**

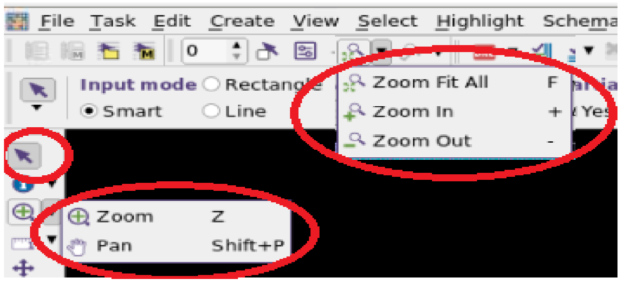
Task 4: Layout Panning and Zooming

✓ Maximize the Layout View by double-clicking on the title bar of the layout tab.

✓ Explore the *zoom* and *pan* buttons as shown below. Note that once you select *zoom* or *pan*, your cursor changes to the magnifying glass or hand icon, and the tool remains in that mode.

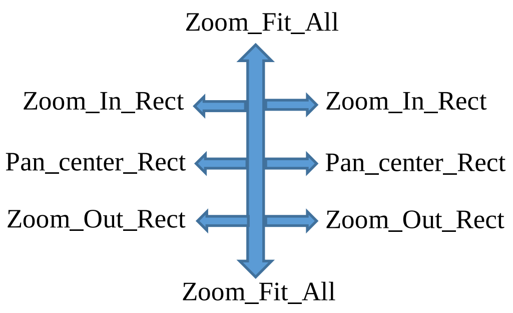
✓ To exit the *zoom* and *pan* mode press the [ESC] key or pick the selection tool (the arrow icon). The cursor returns to an arrow or pointer shape.

✓ Use hot keys: [**F**] or [**Ctrl F]** both correspond to zoom Fit All (or full view), [+] or [=] is Zoom in 2X, [-] is Zoom out 2X.



✓ Use your mouse scroll wheel to zoom in/out (2X or $^1/_2$ X) around the area of the mouse's pointer.

- ✓ Click and hold the middle mouse button or wheel while moving the pointer **straight up or down** and holding it there. The stroke menu appears near the pointer as shown below.



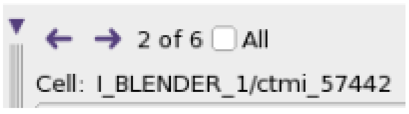
Release the middle button and the design should ZOOM to fit the display window (Zoom Fit All). to zoom in on an area place your cursor in the area of interest then stoke (move the mouse with the middle button or wheel depressed) in a 45^0 direction upward (to the left or right) - the view should zoom in to a rectangular area defined by the stroke.

Task 5: Saving the View Settings

- ✓ To keep the view settings you have made so far, you need to save them. Otherwise, they will revert to the default settings the next time you start the fusion compiler
- ✓ Make sure the View Setting tab (to the right of the Layout tab) is visible and expanded.
- ✓ To save the view settings, click the ☰ "preset" button and select **Save preset as** …
- ✓ Type MyPreset into the present name field, check the **include color and pattern settings box** (to save the new voltage area color setting), and click OK. Fusion Compiler creates a file MyPreset.tcl in the ~/.synopsys_fc_gui/presets/Layout directory. All saved presets will be loaded automatically whenever FC is started again.
- ✓ Select **Default** from the pull-down list of presets to restore the default settings. Fit the view to the window **[F].**
- ✓ Select **MyPreset** to return to your saved view settings.
- ✓ Make one or more changes to your view settings (turn off **Cell** visibility, for example). You can easily return to your *MyPreset* settings by clicking the "***reload***" circular arrow.

Task 6: Querying and selecting Objects

- ✓ To be able to query or select objects, the mouse cursor must be an arrow, which denotes select mode.
- ✓ Hover the pointer over an object without clicking the mouse. The object is highlighted with dashed white lines, and an *Info Tip* box appears in the bottom-left of the Layout view, displaying some key attributes of the object.
- ✓ Select a single object, and hover the mouse cursor over a different object. Notice that the *Info Tip* box displays information about the dashed white line object, not the selected (solid white line) object.
- ✓ To obtain a full query of a selected object press lower-case **[q]** or use the menu: **select** ➜ **Query Selection.** A new Query tab opens below the layout that lists all the attribute values of the selected object.
- ✓ If you query more than one object you can cycle through the individual object by clicking the ⬅ and ➜ arrows at the top of the *Query* panel:

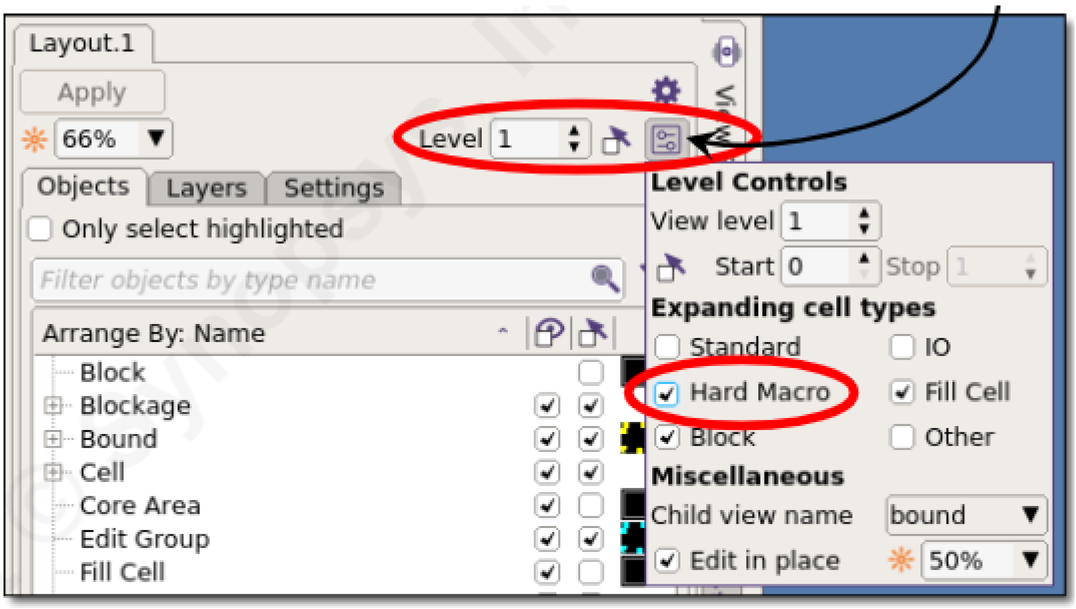


Cell: I_BLENDER_1/ctmi_57442

- ✓ Type **[Ctrl - D]** to deselect all objects.
- ✓ Close the *Query* tab, and make sure the *objects tab of the View Settings* panel is visible on the right.
- ✓ Enable ***Route*** visibility, zoom into a small area, then click on an area with multiple objects stacked on top of each other. Notice that one object will be selected (solid white), while a different object will be queried (dashed white). The info Tip box is associated with the dashed object.
- ✓ 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.
- ✓ Reduce the brightness of the unselected objects, thereby increasing the contrast. Brightness control is located near the top of the view setting panel.
- ✓ Disable Route Visibility and return to 100 % brightness.

Task 7: 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".
- ✓ 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 **Ce**ll, 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, select the black (no fill) pattern, then click **OK**
- ✓ 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**

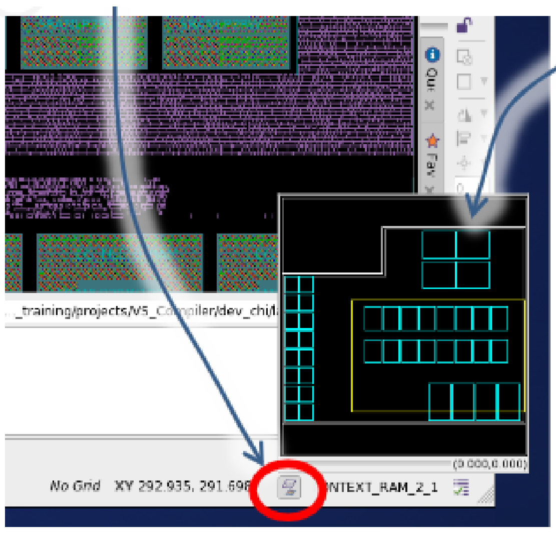


- ✓ If you zoom into one of the RAMS you should see the structures inside the macro. To see what these are, select the small button to the right of the Level setting (Multiple Levels Active). You can turn layer visibilities off/on to see individual layers using the **Layers** tab. Since these are frame views, what you are seeing is mostly routing blockages.
- ✓ Uncheck the *Multiple Levels* Active box and return to a view level of 0

Task 8: Rearranging Panels, Overview

- ✓ You can switch back and forth between the panels (or tabs) on the right (View Settings and Favorites, for example). Sometimes it can be useful to display more than one panel at the same time.

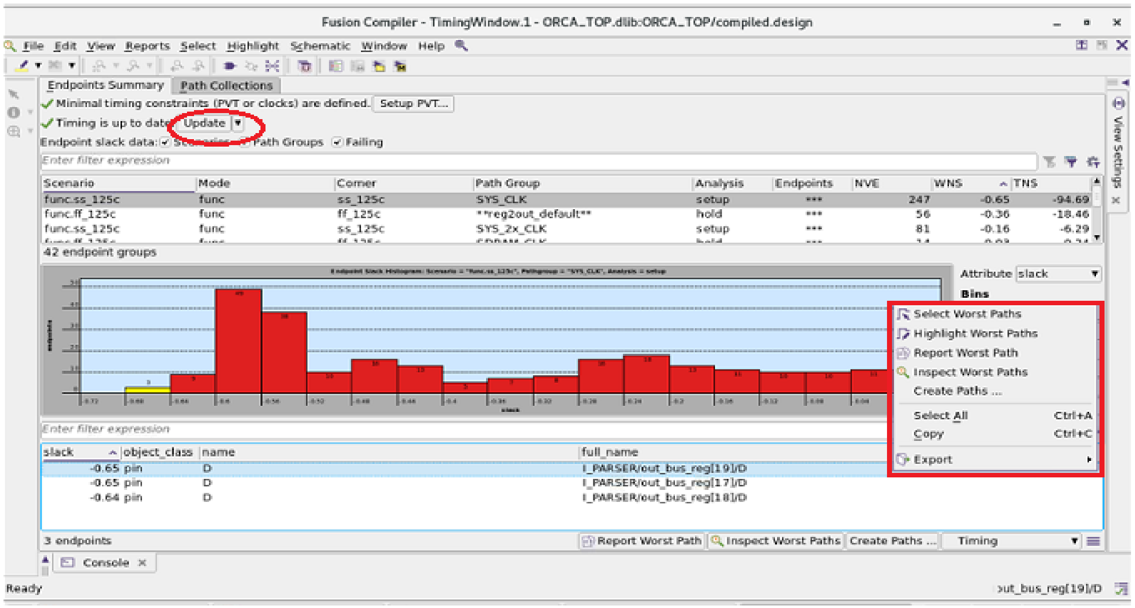
- ✓ Select a macro and press **q** to bring up a query panel.
- ✓ Right-click on the query **tab** - a context menu should be displayed, click on the **Dock 'Query' Right** arrow to move the tab to the right.
- ✓ Right-click on the query **tab** again and in the context menu click on the split 'Query' Below button. The panel area on the right will now spilt into a top and a bottom area, with the query panel being in the bottom area.
- ✓ 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 a later 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.
- ✓ When you right-click on the empty area under or to the right of the tabs, or on a tab (as 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.
- ✓ Try the other menu banner icons, like Detach, Dock, and Collapse. Note that you are not limited to just two-panel areas - you can split the panels again.
- ✓ In the bottom-right of the *BlockWindow*, click on the **Display overview of view** icon (the layout view has to be active)


_training/projects/V5_Compiler/dev_chi/
No Grid XY 292.935, 291.696
NTEXT_RAM_2_1
(0.000,0.000)

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 the yellow 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.

Task 9: Timing Analysis

- ✓ Use the menu **Window** ➔ **Timing Analysis Window**. In the *Timing Window* that appears, press the **Update** button in the top-left to update the timing.
- ✓ Select the scenario func.ss_125c and the path group SYS_2x_CLK.
- ✓ A histogram for the timing distribution is shown for the currently selected scenario By default only violating paths are shown as red bars if there are no bars, there are no violations If you want to see paths that meet timing as green bars, then unselect the **Failing** box above the scenario list (if you do this when done, re-check the **Failing** box and re-select the scenario func. 125 and the path group SYS_2x_CLK).



Fusion Compiler - TimingWindow.1 - ORCA_TOP.dlib:ORCA_TOP/compiled.design

✓ Minimal timing constraints (PVT or clocks) are defined.
✓ Timing is up to date
Endpoint slack data: ✓ Scenarios ✓ Path Groups ✓ Failing

Scenario	Mode	Corner	Path Group	Analysis	Endpoints	NVE	WNS	TNS
func.ss_125c	func	ss_125c	SYS_CLK	setup	***	247	-0.65	-94.69
func.ff_125c	func	ff_125c	**reg2out_default**	hold	***	56	-0.36	-18.46
func.ss_125c	func	ss_125c	SYS_2x_CLK	setup	***	81	-0.16	-6.29

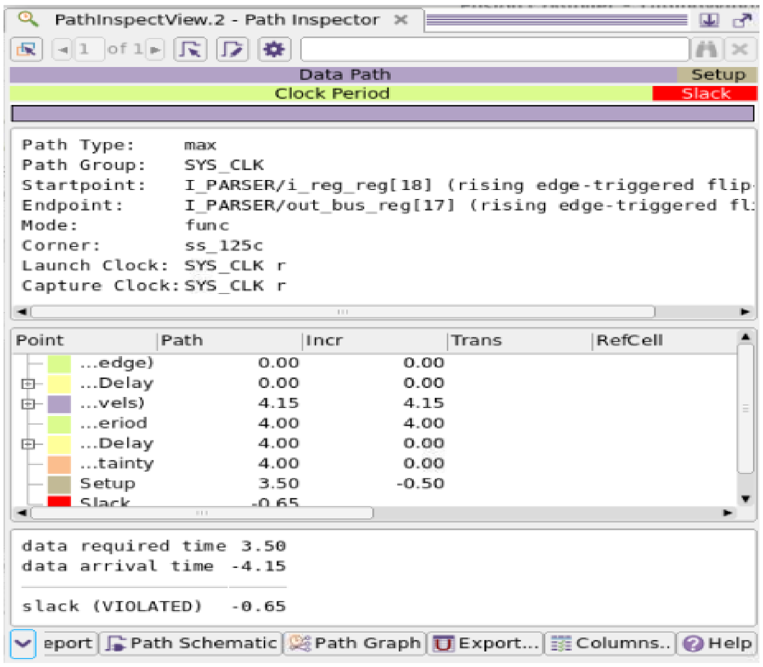
42 endpoint groups

Endpoint Slack Histogram: Scenario = "func.ss_125c", Pathgroup = "SYS_CLK", Analysis = setup

slack	object_class	name	full_name
-0.65	pin	D	I_PARSER/out_bus_reg[19]/D
-0.65	pin	D	I_PARSER/out_bus_reg[17]/D
-0.64	pin	D	I_PARSER/out_bus_reg[18]/D

3 endpoints

- ✓ Click on the left-most red bar (turns yellow) and you should see timing end-points displayed on the bottom
- ✓ Select the first entry, then right-click and choose **"Select Worst Paths"** from the context menu: The timing path is selected and highlighted (white lines and arrows) in the layout view
- ✓ From the *TimingWindow* try the *Path Inspector* by right-clicking a timing path endpoint and choosing **Inspect Worst Paths** from the context menu The reporting format is similar to the output of report_timing. Selecting an item from this window will automatically select the same object in the layout view window



```
Path Type:      max
Path Group:     SYS_CLK
Startpoint:     I_PARSER/i_reg_reg[18] (rising edge-triggered flip
Endpoint:       I_PARSER/out_bus_reg[17] (rising edge-triggered fl
Mode:           func
Corner:         ss_125c
Launch Clock:   SYS_CLK r
Capture Clock:  SYS_CLK r
```

Point	Path	Incr	Trans	RefCell
...edge)	0.00	0.00		
...Delay	0.00	0.00		
...vels)	4.15	4.15		
...eriod	4.00	4.00		
...Delay	4.00	0.00		
...tainty	4.00	0.00		
Setup	3.50	-0.50		
Slack	-0.65			

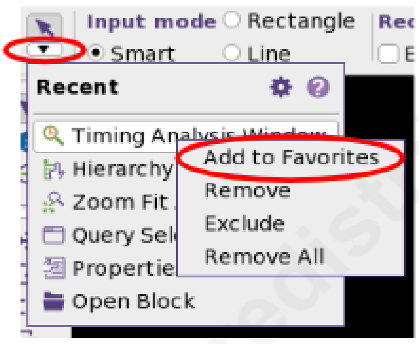
```
data required time  3.50
data arrival time  -4.15

slack (VIOLATED)   -0.65
```

- ✓ Close the *Timing Window* (containing the Timing Status, PathInspector, and SourceBrowser tabs).

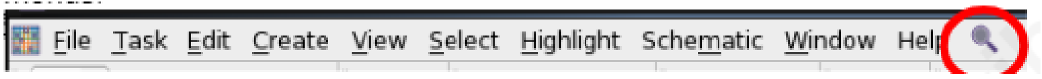
Task 10: Using the Recent menu and Favorites

- ✓ Bring the *BlockWindow* to the foreground.
- ✓ Whenever you perform a GUI function, like 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.
- ✓ You will see this function listed in the *Favorites* pane.



Task 11: Getting Help

- ✓ 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 *BlockWindow*, on the right end of the menus.



- ✓ In the "Search for commands" field, begin typing compile. Notice that as you type, it lists all the menus, commands, and application options that relate to that search string.
- ✓ Select one of the **Application Options**, for example: Compile.flow.enable_ccd
- ✓ 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,
- ✓ Application options are used to configure various aspects of how the *Fusion Compiler* works. You will learn more about them during the workshop.
- ✓ *Fusion Compiler* supports the command, variable, file name, and command option completion through the [**Tab**] key.
- ✓ Try the following in the console at the fc_shell> prompt:

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

- ✓ 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_optionsfor application options, and use report_app_var for variables, along with the *wildcard. Here are some examples:
- ✓ Let's say you are looking for more information about the command to modify GUI hotkeys, but you do not remember the exact command name. You know it contains the string "key". To list all commands that contain this string enter: `help *key*`
- ✓ From the displayed list of commands, you find the one you are interested in, namely, qui_set_hotkey.
- ✓ Of course, you could have entered the key in *Command Search* as well.
- ✓ The above command only searches through all command names. if you want to search through command options as well, use the following: `apropos key`
- ✓ This will return all commands that have the substring "key", as well as all commands with options (or *man* page descriptions) that match "*key*".
- ✓ To list the available options for a command, for example, gui_set_hotkey.

```
gui_set_hotkey  -[Tab]      or,
help gui_set_hotkey -verbose  or,
help  gui_set_hotkey  -v      or,
gui_set_hotkey     -help
```

- ✓ To get a full help *manual* page - a detailed description of the command and all of its options. type

```
man gui [Tab]_set[Tab] ho[Tab]       or,
man qui_set hotkey
```

- ✓ 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 *compile*. To list all application options that start with "compile", enter:

```
report_app_options compile*
```

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

```
man compile .flow.enable_ccd
```

- ✓ You can also get additional help for an error or warning message, using the unique message code, for example, man SQM-1034
- ✓ Finally, specifically for this lab, 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 gate
- ✓ The aa function searches through commands, application options, and variables that match the given string. In addition, it shows the current value.
- ✓ Use the -v (verbose) option to also search through apropos.
- ✓ If you want to redirect content to a separate window, try this: v aa gui
- ✓ **v** is a user-defined alias that calls the "view" custom function, useful for viewing long reports, and allows for regular expression searches.
- ✓ To list the workshop-provided helper functions and aliases, type "ces_help".
- ✓ Quit *Fusion Compiler* by using the menu **File** ➔ **E**xit, or by typing *exit* at the command prompt. In the exit dialog, you can choose if you want preferences and/or window settings to be saved, and if you want to disable this exit window confirmation dialog from popping up the next time.

Questions

1. List and explain files, which are created while invoking FC.
2. What is the Hotkey to Open Block?
3. What is the boundary shape of the block?
4. What is the color of the Macros and Standard cells?
5. What is the utilization rate of the ORCA_TOP design?
6. Find out the number of standard cells and Macro present in the design.
7. How many sub-module and hard macros are present in the I_SDRAM_TOP module?
8. List out the Hotkeys definitions from the pull-down menu **Help➔ Report Hotkey Bindings.**

Lab 02 - Reading RTL

Objective : *Familiarize yourself with the process of reading RTL into Fusion Compiler.*

Working Directory : /home/<user_name>/PD_Labs/FC/lab02

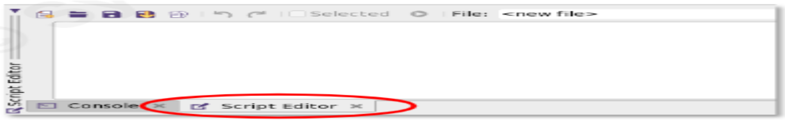
Project Name : ORCA_TOP.dlib

Learning outcomes :

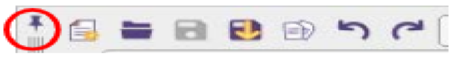
- ✓ Create a design library
- ✓ Read RTL files
- ✓ Debug some common library setup mistakes
- ✓ Use design mismatch management (DMM) to read dirty RTL
- ✓ Save the block

Task 1: Directory Structure and Invoking FC

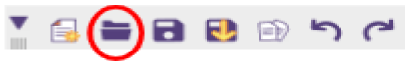
- ✓ Change to the lab directory to *lab_2*, then invoke FC using: fc_shell -gui
- ✓ Look at the Console and Script Editor tabs in the bottom-left of your GUI window. If you see the Expand (up-arrow)icon, the Console window is collapsed. Expand it by clicking on the **Expand** icon (if you do not see the up-arrow icon, the window should already be expanded)
- ✓ Select the Script Editor tab to open the script editor window:



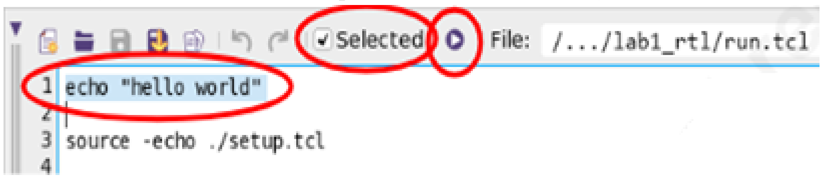
- ✓ If the Script Editor window is "unpinned", you will see the "pin" icon shown below. To keep the Script Editor window from auto-closing when you make a different GUI "active", click on the **pin** icon.



- ✓ Now click the Open (folder) button, then select and Open the file run.tc1.



- ✓ 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** (play) icon.
- ✓ Try this now: Double-click to select the entire line echo "hello world", ensure that **Selected** is checked, then click the **Run** button. Look at the results in the shell window or by selecting the *Console tab* (switch back to the *script Editor* tab when done).



```
echo "hello world"

source -echo ./setup.tcl
```

Task 2: Create the Design Library, Read the RTL

- ✓ Open a new terminal window and change your current directory to **lab_2**. examine the files that will be used in this lab, and take a quick look at the **setup.tcl** file:

- ✓ This file defines the **search_path** application variable as well as user-defined variables which are used by commands in the **run.tcl**. Additionally, it specifies the number of threads to use and suppresses unwanted messages.
- ✓ Close the **setup.tcl** file — do not save it.
- ✓ From the run.tc1 file in the script editor, select and run the command to load the setup file:

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

- ✓ Select and run the command to create the design library:

```
Create_lib \
-technology $TECH_ FILE
-ref_libs$REFERENCE_LIERARY \
risc_core.dlib
```

- **Question 1**. What is the name of the newly-created design library?

  ……………………………………………………………………………..
  ……………………………………………………………………………..
  ……………………………………………………………………………..
- **Question 2**. Did the design library show up in the **lab_2** directory?

  ……………………………………………………………………………..
  ……………………………………………………………………………..
  ……………………………………………………………………………..

- ✓ Analyze the Verilog RTL files:

```
analyze -format verilog [glob risc_rtl1/*.v]
```

  - Note: The glob command used above returns a list of all files that match the pattern.
- ✓ Create a linked "top" block:

```
elaborate risc_core
set_top_module risc_core
```

- ✓ The *BlockWindow* in the GUI shows the layout of the **RISC_CORE** block. Currently, only standard cells (pink-purple rectangles) are shown, stacked on top of each other in the lower-left origin.
- ✓ In the shell window or *Console* tab you should see several warnings about issues in the linking process, which leads to a failed link.
  - Error: Block'risc_core' failed to properly link during 'set_top_module'. (DES-220)
- ✓ Let's first focus on the following issue:
  - Warning: unable to resolve reference to 'SRAMLP2RW128x16' first referenced from module 'reg_file_use_rams1'. (LNK-005)
- ✓ The SRAMLP2RW128x16 component is an SRAM hard macro. This and other SRAMs are defined in a separate cell library which should have been specified as one of the reference libraries (create_lib -ref_libs ...), but it was not.

  - **Question 3.** What is the name and location of the reference library containing the unresolved references? (HINT: Look at the user-defined variables in the setup.tcl)

    ……………………………………………………………………………..
    ……………………………………………………………………………..
    ……………………………………………………………………………..
- ✓ This 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.tc1. If the 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 the design setup needs to be repeated, you will not encounter the same error.

- ✓ **You will execute both methods to fix the problem.**

Method #1

- ✓ Add the missing reference library to the existing list using set_reg_libs, then re-link:

```
Set_ref_libs  -add ../ref/CLIBs/saed32_ sram_lp.ndm
Link_block  -force
```

- ✓ The GUI now includes the SRAMs (large blue-green rectangles) stacked on top of the much smaller standard cells.
- ✓ The design should *link* without the warning about the SRAM component (the other errors should still be there). Note that we used **link_block** to re-link the block instead of **set_top_module**, which can only be executed once per session. To force the link process, it is therefore, in this situation, necessary to use **link_block -force**.
- ✓ Note also that the final log message says **"Design 'risc_core' was successfully linked."**, even though there are linking errors! A"successful link" allows you to continue with the flow if you choose: For example, you may decide to tum unresolved instances into black boxes.
- ✓ 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.

Method #2

- ✓ **Edit** setup.tcl and add the missing reference library to the REFERENCE_LIBRARY list
- ✓ Close the design library, then repeat the main design setup steps:

```
close lib
Source -echo ./setup.tcl
Create_lib -technology$TECH_FILE \
 -ref_libs$REFERENCE_LIBRARY \
risc_core.dlib
analyze -format verilog [glob risc_rtl/*.v]
elaborate risc_core
Set_top_module risc_core
```

- ✓ Confirm that the SRAM CLIB has been added as a reference library:`report_ref_libs`
- ✓ Examine the remaining Warnings:

  Warning: Unable to resolve reference to 'alu' first referenced from module 'risc_core'. (LNK-005)

  Warning: Cannot find port "half_full' on reference 'control', referenced by instance 'control'. (LNK-008)

  Warning: Unable to resolve reference to 'control' first referenced from module 'risc_core'. (LNK-005)

  Warning: Cannot find port 'pc' on reference 'prgrm_cnt_top', referenced by instance 'prgrm_cnt_top'. (LNK-008)

  Warning: Unable to resolve reference to 'prgrm_cnt_top' first referenced from module 'risc_core'. (LNK-005)

- ✓ There are three unresolved references, two of which might be related to the warnings on missing ports. The first warning is because the ALU module is missing in the RTL files in the **risc_rtl** directory:
- ✓ In a Linux window confirm that there is no module for alu:

```
% grep module risc_rtl/*
```

- ✓ Because there are RTL problems ("dirty" RTL), you will not be able to link the design successfully with this current setup. You will use *design mismatch management* (DMM) in the next task to address the issues.

Task 3: Use DMM for Dirty Netlist Handling

- ✓ In this task, you will use Fusion Compiler's *design mismatch management* (DMM)capabilities to read dirty RTL for a successful link.
- ✓ Use copy/paste or select / Run within the script editor.
- ✓ To remove the design that has been read so far, the easiest way is to simply close the current block:close_blocks
- ✓ As you will see from the error message, Fusion Compiler prevents the accidental closing of modified blocks. Use the **-force** option: close blocks -force
- ✓ Set up DMM as you have learned in the lecture:

```
Set_current_mismatch_config auto_fix
get_current_mismatch_config
```

- ✓ Re-read the RTL, this time using a VCS file list. The risc_core.vcs file contains VCS instructions to read the verilog files, and can be used without modification with the **analyze** command.

```
analyze -vcs "-f risc_core.vcs"
elaborate risc_core
Set_top_module risc_core
```

- ✓ As you will see, the linking is now successful (no more warnings or errors). You will also see that several repairs have been performed.

Information: Disconnected pin 'half_full' of instance 'control', in module 'risc_core' (LNK-045)
Information: Removed pin 'half_full' of instance 'control', in module 'risc_core' (LNK-046)
Information: Renamed pin 'pc[7]' of instance 'prgrm_cnt_top', to 'pc[7]', in module 'risc_core' (LNK-047)
Information: A total of 9 mismatches are found on block 'risc_core.dlib:risc_core.design'. (DMM-116)
Committed cell alu to blackbox design alu

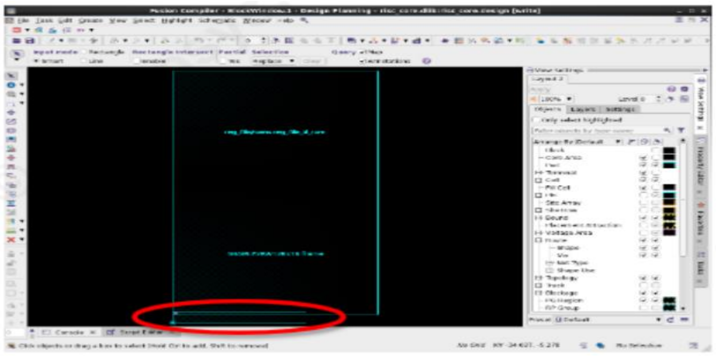
- ✓ Find out more information on some of the repairs that were performed:

```
report_design_mismatch  -verbose
```

- ✓ This will not report all the fixes that were made (in our case, the pin disconnection/removal is not). It does show the creation of a black box for missing module alu as well as the port name case mismatch.
- ✓ Under "normal" circumstances all top-level references are defined either by a module/entity in the RTL, or a cell in a CLIB. Since, by definition, a black box does not exist before its creation, and *Fusion Compiler* cannot modify the RTL or CLIBs, *Fusion Compiler's* only choice is to create a new block for the missing module, which will reside in the design library. In our example the block will be called alu.design: get_blocks (risc_core.dlib: risc_core.design risc_core.dlib: alu.design)
- ✓ Find out the size of the black box boundary that was automatically assigned:

```
get_attribute [get_blocks alu.design] boundary
```

- ✓ The alu black box appears as a separate rectangle in the lower-left of the layout.



- ✓ Change the boundary shape of the alu black box by running theset_attribute command as shown

```
set_attribute [get_blocks alu] \
boundary {{0 0} {0 160} {120 160} [120 0}}
```

- ✓ The change will be reflected in the GUI immediately.

Task 4: Save the Block

- ✓ You have completed the first part of the design setup steps; You will complete the remaining design setup steps as well as *timing* setup in a later lab. This is a good time to save the block.
- ✓ List the files and directories in the current directory **lab_2**: ls -l

  - **Question 4.** Does the **risc_core.dlib** design library exists in the current directory?

    …………………………………………………………………………..
    …………………………………………………………………………..
    …………………………………………………………………………..

- ✓ Execute the following commands to rename the block with the label read which helps to identify its current state, and then to save both the**risc_core** and **alu** (black box) blocks and design library:

```
rename_block -to_block risc_core/read
save_lib
```

  - **Note:** Using save_block instead of save_1ib would have saved only the current block, risc_core, instead of both blocks.

  - **Question 5**. Does the risc_core.dlib design library show up now?

    …………………………………………………………………………..
    …………………………………………………………………………..
    …………………………………………………………………………..

- ✓ To explore the block naming convention, the default block handle format is libName:blockName.viewName.
- ✓ The default viewName is design. As was done in the previous step, a block can be saved with an optional label, in which case the block handle is:libName:blockName/labelName. viewName.
- ✓ lf a design library contains multiple blocks with the same **block name** (but with a different label name and/or viewName), the block must be referred to by its unique block handle. lf the design library contains only one block with a certain **block name** (as is the case with the **risc_core** block), then that block can be referred to simply by its block Name . You can optionally include the libName, and/or labelName, and/or viewName, as demonstrated by executing these commands:

```
get_blocks risc_core/read
get_blocks risc_core/read.design
get_blocks risc_core.dlib:risc_core/read
get_blocks risc_core.dlib:risc_core/read.design
```

- ✓ Exit out of Fusion Compiler. The GUI will ask for confirmation — click **Exit:** exit

Lab 03 - Compile Flow

Objective : *Explore the 7 stages of the compile_fusion process.*

Working Directory : /home/<user_name>/PD_Labs/FC/lab03

Project Name : ORCA_TOP.dlib

Learning outcomes:

- ✓ Read the RTL and link the design
- ✓ Apply technology setup
- ✓ Apply the UPF power intent
- ✓ Create the modes/corners/scenarios and apply the timing constraints
- ✓ Apply auto-floorplan settings

Task 1: Read the RTL, Complete the setup.

- ✓ Invoke Fusion Compiler from the lab_3 directory: `fc_shell -gui`
- ✓ In the GUI, click on the button Execute script
- ✓ In the file dialog that opens, select the file **setup.tcl**, then **Open**.
- ✓ This will perform all the steps outlined above. Take a quick look at the file, but don't spend too much time on this since we will be covering most of these steps in later labs.
- ✓ From the *Script Edit*or bottom tab select **Open** (folder), then if needed use the**Parent Directory up-arrow** to **Open** the lab_3 directory, and the o**pen** run.tcl
- ✓ This file contains all the commands that you will be executing in this lab.
- ✓ Check the Selected box, select command(s) from this file, and press instead of typing them yourself, to save time and avoid typos.

Task 2: Run compile_fusion initial_map

- ✓ Run **stage 1** of compile, *initial_map:*

```
_run_compile_stage_save initial_map
```

  - **Note:** _run_compile_stage_save is a procedure that runs an individual compile stage, creates a log file for the executed stage, then saves the block under that stage name. It is a little simpler to enter compared to compile fusion -from <stage> -to <stage>save_block -as <stage>This is not a native command to Fusion Compiler.
- ✓ Have a look at the log file compile_fusion.initial_map.log. You should be able to observe that:
  - MV cells are inserted (if necessary - here, none are required)
  - Shift registers are identified
  - Unloaded registers are removed
- ✓ Report the unloaded registers (including removed registers):

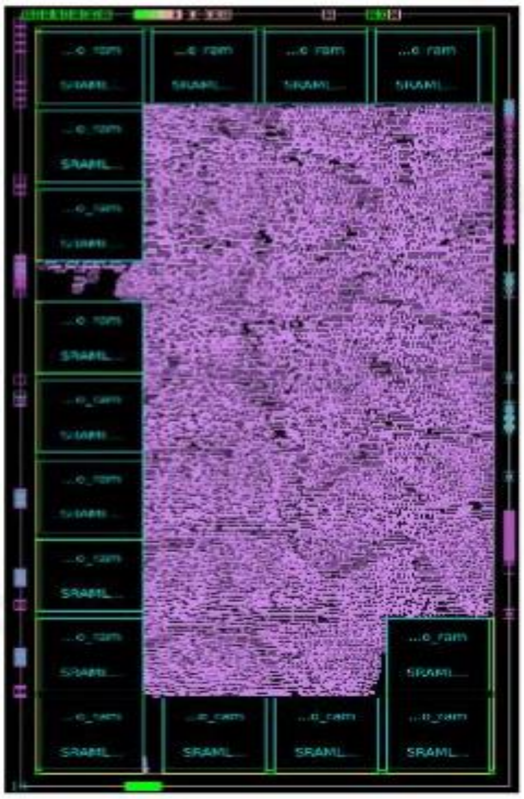
```
report_unloaded registers
```

- ✓ ICG insertion also occurs during initial_map. Verify that 5 ICGs were inserted:

```
report clock_gating
```

Task 3: Run compile_fusion logic_opto

- ✓ Run **stage 2** of compile, logic_opto:`_run_compile_stage_save logic_opto`
- ✓ You should see that the floorplan has been created automatically, pins are placed along all 4 edges, and a first coarse placement has taken place (zoom in to examine). Timing-driven logic optimization occurs before and after placement. Here is a screenshot of the design after stage 2:



- ✓ Create a test protocol for specified clock signals and control signals.

```
create_test_protocol
```

- ✓ Here, the test protocol is created for two clocks namely **pclk** and **sys_clk**and the two asynchronous control ports namely pci_rst_n andsys_reset. For the user specifications, please refer **pci_dft.tcl** which is present in the script directory.
- ✓ Checks the current design against test design rules: `dft_drc -v`
- ✓ You should get DRC summary report without any violations.
- ✓ Enable the use of insert_dft in the breakpoint flow, after the compile_fusion-to logic_opto is performed.

```
set_app_option-name dft.insertion_post_logic_opto  -value true
```

- ✓ Insert the DFT structures in the current design and save it.

```
Insert_dft
Save_block
```

Task 4: Run compile_fusion initial_place

- ✓ Run **stage 3** of compile, initial_place:`_run_compile_stage_save initial_place`
- ✓ The placement performed here is buffering-aware timing-driven placement(confirm by searching for the second "Placement Options" table in the log file (**compile _fusion.initial_place.log**).
- ✓ Note that the placement is still coarse, the standard cells are not legalized.
- ✓ Find high-fanout nets:

```
all_high_transitive_fanout  -nets  -threshold 100
```

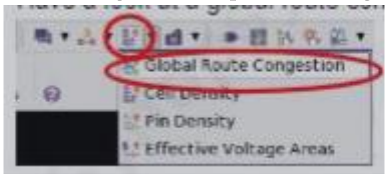
- ✓ You should see the reset net **pci_rst_n**. This will be buffered during thenext stage. The other net is the output net of an integrated clock-gating cell(IGG) which will be buffered during clock tree synthesis (CTS).
- ✓ Report design statistics:`report_design`
- ✓ From the Count column note down the number of:
  - Standard cells: ____________
  - Buffer/inverter: ____________

Task 5: Run compile_fusion initial_drc

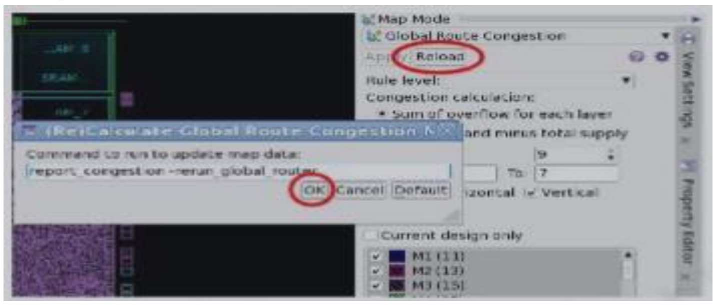
- ✓ Run **stage 4** of compile, initial_drc: _run_compile_stage_save initial_drc
- ✓ Run report_design, and note down the number of
  - Standard cells: ____________
  - Buffer/inverter: ____________
- ✓ Notice that the standard cell count increase is primarily due to DRC buffering.
- ✓ Confirm that the reset net pci_rst_n has been buffered:

```
all_high_transitive_fanout    -nets   -threshold 100
all_high_transitive_fanout    -nets   -threshold 40
```

- ✓ The *clk nets are clock nets, which will be buffered later during CTS.
- ✓ Find out how many scan chains there are:get_scan_chain_count
- ✓ Have a look at the global route congestion map of the design by first selecting:



- ✓ In the new panel that appears, select **Reload** then **OK** to run global routing and display the congestion map:



```
report_congestion -rerun_global_router
```

- ✓ you should see that there are no bright green, yellow or red "hotspots", which indicates that there is hardly any congestion (discussed in a later Unit).
- ✓ Close the congestion map by clicking on the "**X**" on the **Map Mode** panel **tab** on the right of the GUI.

Task 6: Run compile_fusion initial_opto

- ✓ Run **stage 5** of compile, initial_opto:_run_compile_stage_save initial_opto
- ✓ This stage performs many optimization steps, including CCD, scan insertion, etc.
- ✓ Zoom into the layout and you will see that placement seems to be fully legalized -standard cells are placed inside the placement rows and are not overlapping. Run the following command to check whether all cells are indeed placed legally:check_legality
- ✓ You will find that this is not the case: Several cells are overlapping. Some remaining illegal placement is expected at this stage of the design. Final legal placement will be accomplished during the last compile stage.
- ✓ Verify that the design is already close to meeting timing. Even though this is a simple test case, a lot of timing optimization has occurred during this stage: rt
- ✓ At any time, even after closing and reopening the design, you can find out what compile stages have been run. Execute the following command:report_optimization_history
- ✓ This shows the stages that were run, the time of execution as well as the runtime of each stage.

Task 7: Run compile_fusion final_place

- ✓ Run **stage 6** of compile, final_place:`_run_compile_stage_save final_place`
- ✓ Check placement legality again:`check_legality`
- ✓ This time cell placement is fully legal. This is not the last stage though, more optimizations and incremental placement will occur during the 7th compile stage, after which final legalization takes place.
- ✓ Review the log file **compile_fusion.final_place.log**
- ✓ Run a timing summary:`report_qor -summary`
- ✓ You should see that there are a few timing as well as max tran/cap violations due to the finalized placement, which the next stage should improve.
- ✓ Report the power consumption of the design:`report_power`
- ✓ Note down the Total Internal and the Total Leakage Power:
  - Total Internal Power: __________
  - Total Leakage Power: __________

Task 8: Run compile_fusion final_opto

- ✓ Run **stage 7** of compile, final_opto:`_run_compile_stage_save final_opto`
- ✓ You should see a significant improvement in leakage power:`report_power`
  - Total Internal Power: __________
  - Total Leakage Power: __________
- ✓ Confirm that there are no (or very few and very small) timing violations left:

```
report_qor -summary
```

- ✓ Exit out of Fusion Compiler by entering: `exit`
- ✓ then selecting the following in the GUI: **Discard All** ➔ **Close Block** ➔ **Exit**

Lab 04 - Technology, UPF, Floorplan, and Timing Setup

Objective : *To get familiarized with the design setup steps such as loading UPF, floorplan, and timing setup.*

Working Directory : /home/<user_name>/PD_Labs/FC/lab04

Project Name : ORCA_TOP.dlib

Learning outcomes:

- ✓ Load UPF
- ✓ Load a floorplan
- ✓ Check and modify the placement site and routing layer settings
- ✓ Load MCMM timing constraints
- ✓ Check and correct timing setup issues

Task 1: Create a copy of Block from Lab02

This lab is a continuation of lab_2 where the RTL design was read. At the end of lab_2, the block was saved in the **risc_core.dlib** design library as risc_core/read. A completed version of that block has been provided here so that this lab does not rely on the completion of lab_2.

- ✓ Invoke Fusion compiler from the lab directory:

```
Unix%  cd  lab_4
Unix%   fc_shell -gui
```

- ✓ From the script, the editor tab selects Open (folder) and uses the Parent Directoryup-arrow to open the lab_4 directory. then open run.tcl. 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.
- ✓ Source the **setup.tcl** file to ensure that the search_path is set up:

```
source –echo setup.tcl
```

- ✓ Copy risc_core/read to risc_core/setup:

```
open_lib risc_core.dlib
copy_block –from risc_core/read –to risc_core/setup
current_block risc_core/setup
```

Task 2: Technology Setup

- ✓ Before you compile the design you need to perform a technology-specific setup, which includes, but is not limited to.
  - Loading RC parasitic models
  - Specifying site symmetry and default site attribute values
  - Defining preferred metal routing directions
  - Specifying ignored metal routing layers.
- ✓ Load the TLUPlus RC parasitic interconnect models needed for optimization:

```
read_parasitic_tech \
-layermap ../ref/tech/saed32nm_tf_itf_tluplus.map \
-tlup ../ref/tech/saed32nm_lp9m_Cmax.1v.nxtgrd \
-name maxTLU

read_parasitic_tech \
-layermap ../ref/tech/saed32nm_tf_itf_tluplus.map \
-tlup ../ref/tech/saed32nm_1p9m_Cmin.lv.nxtgrd \
-name minTLU
```

- ✓ Confirm that the models have been properly set up:

```
report_lib –parasitic_tech risc_core.dlib
```

- ✓ Get a list of the defined site definitions:`get_site_defs`
- ✓ Set the symmetry attribute for the site definition **unit** to **Y-symmetry**

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

- **Question1.**Does Y-symmetry mean that standard cells can be flipped in the Y-direction (along the X-axis) or flipped in the X-direction (along the Y-axis)?
  …………………………………………………………………………
  …………………………………………………………………………
  …………………………………………………………………………

- ✓ Set the default site definition:

```
set_attribute [get_site_defs unit] is_default true
```

- ✓ Set the metal layer preferred routing directions then confirm:

```
set_attribute [get_layers {M1 M3 M5 M7 M9}] routing_direction horizontal
set_attribute [get_layers {M2 M4 M6 M8}] routing_direction vertical
get_attribute [get_layers M?] routing_direction
```

- **Note:** The get_layers M? command returns the following collection {M1 M2 M3 M4M5 M6 M7 M8 M9}, so the list of "horizontal vertical horizontal …" values of the routing direction attribute follow this layer order.

- ✓ Confirm that all metal layers are available for signal routing (none are ignored), by default, then limit routing to M8 or lower:

```
report_ignored_layers
set_ignored_layers -max_routing_layer M8
report_ignored_layers
```

Task 3: Load UPF

- ✓ Take a quick look at the UPF file located at Design_**data/risc_core.upf**. This is a simple UPF file that defines:
  - The power supply nets/ports: VDD and VSS
  - The power domain PD_TOP
  - The power states of the power nets: Only define an ON-state here
- ✓ Close the UPF file — do not save it - then load and commit the UPF:

```
load_upf risc_core.upf
commit_upf
```

- ✓ It is good practice to perform a multi-voltage design check after loading UPF:

```
check mv design
```

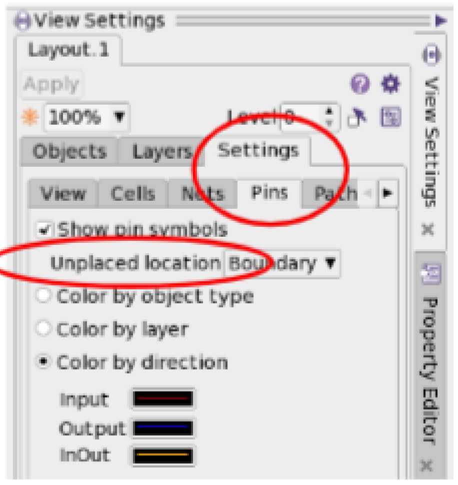
  The only warnings you should see are about tie-off connections that have not been implemented yet. This is not an issue since tie-offs will be implemented during compilation.
- ✓ Use the GUI to visualize the UPF you have applied.
- ✓ Use the menu View➔ Multivoltage Views ➔ **Supply** Network. This is a useful tool to help debug your UPF.
- ✓ Therearenoissueswithoursimple UPFfile.so this step is only intended to let you know about this feature. Try the other entries of the **Multivoltage Views** menu as well but do not spend too much time here. since this is beyond the scope of this lab.

Task 4: Load the Floorplan / Make Tie - Cells Available

- ✓ For purposes of this lab, we are assuming that the RTL was already compiled previously with a default floorplan, to generate a netlist used only to create the floorplan. In Lab_2 and in this lab we have been performing design setup (again) to include the actual floorplan.
- ✓ Load the floorplan generated by floor planning:

```
source design_data/risc_core.fp/floorplan.tcl
```

- ✓ You will see two warnings about "_dummynet0/1". These are two nets that connect to the **scan_en** and **test_mode** ports in the DEF file, but they don't exist in your design yet.
- ✓ Look at the Layout window (**View→ Layout View**) in the GUI Block Window. (close the UPF view if still open), and you will see the rectangular-shaped floorplan of the **risc_core** block:
- ✓ All the macros, including the **alu** black box, are placed inside the core.
- ✓ Logical ports (square shapes containing the letter "I" or "0) and corresponding physical terminals (metal shapes) for the I/0's and P/G are placed around the block boundary (you must zoom in to see the terminals). Ports are colored by the metal layer of the corresponding terminal by default.
- ✓ If you want to color the ports in their port direction, make the change in **view settings** as shown here
- ✓ A voltage area, called **DEFAULT_VA** is defined for the entire core area (in the Objects tab of view Settings panels on the right **check** the **Voltage Area visibility** box to see the dashed yellow outline of the voltage area)



✓ Show pin symbols

Unplaced location: Boundary

Color by object type / Color by layer / [✓ Color by direction]

Input ▬
Output ▬
InOut ▬

- ✓ Most technologies require tie-cells to be inserted for tie-off connections. If you are not familiar with a library. you can find tie-off library cells by using the function_id attribute.
- ✓ For tie-high cells. this attribute has a value of a0.0.
- ✓ For tie-low cells. this attribute has a value of Ia0.0.

```
get_lib_cells –filter "function_id==a0.0"
get_lib_cells –filter "function_id==Ia0.0"
```

- ✓ For the tie-cells to be used by optimization. remove any **dont_touch** settings, and ensure the **optimization** library cell purpose is enabled:

```
set_dont_touch [get_lib_cells */TIE*] false
set_lib_cell_purpose –include optimization [get_lib_cells */TIE*]
```

Task 5: Multi-Corner Multi-Mode Setup

- ✓ In a different terminal window, open the file: **Design_data/mcmm_risc_core.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 array Name (var Name)) value) to create efficiently-coded for each loop.

  - **Question 2**. How many modes, corners, and scenarios will be created?
    Modes:………………….
    Corners:…………………
    Scenarios:………………..

- ✓ The next section of the **mcmm_risc_core.tcl** sources the mode-, Corner- and scenario-specific constraints files into their respective newly- created modes. corners and scenarios.
- ✓ The constraints files are in **design_data/.** If you have the time, take a quick look at one of each of the modes (_m _). Corner (_c_) and scenario (_s _) files.
- ✓ The last section of the **mcmm_risc_core.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 questors:

- **Question 3.** Which scenarios will be active?
  …………………………………………………………………………
  …………………………………………………………………………
  …………………………………………………………………………

- **Question 4.** Which analysis types will be enabled for the test.ss_125c scenario?
  …………………………………………………………………………
  …………………………………………………………………………
  …………………………………………………………………………

✓ Close the **mcmm_risc_core.tcl** file –do not save it, then source it:

```
source –echo mcmm_risc_core.tcl
```

✓ Generate a scenario report to verify that the scenarios are configured as expected:

```
report_scenario
```

- **Note:** There are two additional analysis types called Cell EM and Signal EM (Electromigration), which are off by default.

✓ Ensure that there are no propagated clocks prior to synthesis:

```
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]
```

✓ Verify that the current mode and corner are consistent With the current scenario:

```
current_mode
current_corner
current_scenario
```

✓ Change the *current_mode* and *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.

✓ Return to the original current scenario when done:

```
current_corner ff_m40c
current_scenario
current_mode test
current_scenario

current_scenario func.ff_m40c
current_mode
current_corner

current_scenario func.ss_m40c
```

✓ 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: true | false Default: false   Empty: false

✓ Current refers to whether this mode is the current mode or not. Default is only true if you did not create any modes on your own. in that case Fusion Compiler would have a single mode named default, and its status would be Default: true. Empty is true if you have not applied any constraints to this mode.

✓ **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
```

✓ Look at the summary section for each corner (between the dashed lines) as well as the detailed sections and answer the following questions:

- **Question 5:** Which corner{s) have PVT mismatches?
  …………………………………………………………………………………
  …………………………………………………………………………………
  …………………………………………………………………………………

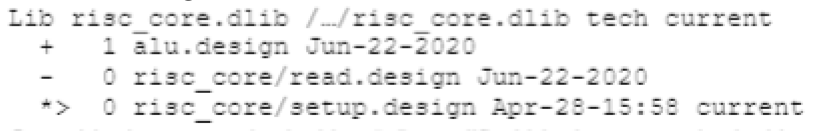
- **Question 6:** What is mismatching — process, voltage, and/or temperature?
  …………………………………………………………………………………
  …………………………………………………………………………………
  …………………………………………………………………………………

✓ 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 7:** What is causing the mismatches?
  …………………………………………………………………………………
  …………………………………………………………………………………
  …………………………………………………………………………………

✓ Based on the name of the corner with the mismatches (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.
✓ Close the PVT report view window.
✓ Make the necessary correction in the appropriate corner constraints file located in the design_data/directory.
- **Note:** Check the answer section at the end of the lab if you need help.

✓ Re-execute the commands in steps 4 and 10 on the previous pages, until a clean PVT report is obtained. Note that if you still see the issues regarding wvgtech you can continue to safely ignore them.
✓ Execute the following to list the blocks:list_blocks
✓ You should see the following.



```
Lib risc_core.dlib /…/risc_core.dlib tech current
  +   1 alu.design Jun-22-2020
  -   0 risc_core/read.design Jun-22-2020
  *>  0 risc_core/setup.design Apr-28-15:58 current
```

✓ Open blocks are marked with a "+", and modified blocks are marked with a "*". Designs are tagged with their modification date or last saved date. The current block is tagged "Current." Designs selected to resolve references by the *link_block* command are marked with a ">"
✓ Save the block: Save_block
✓ Execute**list_blocks** again to confirm that the block has been saved.
✓ Exit Fusion Compiler: exit

Lab 05 - Concurrent Clock and Data (CCD) Optimization

Objective : *Introduce the CCD capabilities in Fusion Compiler*

Working Directory : /home/<user_name>/PD_LABS/FC/lab05/

Project Name : ORCA_TOP.dlib

Learning outcomes :

- ✓ Set up the useful skew application options
- ✓ Analyze and recognize the latency adjustments made by CCD during compile_fusion

Task 1: Load the design and analyze RAM timing

- ✓ Change to the **lab_5** directory and invoke the Fusion Compiler front-end.

```
cd lab_5_ccd
fc_shell  -gui
```

- ✓ Use the script editor and open the **run.tcl** file from the lab_5 directory, or use **“vv run.tcl”** (front of the Fusion Compiler shell Window)to open the file in a view window. The file contains the commands which you will be executing in this lab. You may copy & paste these commands into the fc_shell terminal.
- ✓ Since useful skew computation occurs at the **initial_opto** stage. our starting point for this lab will be a design that has been compiled through the **initial_drc** stage. Open the design library copy the**initial_drc** block to a new block with the label initial_opto and make it the current block: Also, increase the default reporting resolutions

```
open_lib risc_core.dlib
copy_block -from risc_core/initial_drc \
  -to risc_core/initial_opto
current_block risc_core/initial_opto
set_app_options –list {
shell.common.report_default_significant_digits 3
```

  - Note down the latencies and uncertainties of the clocks in this design

```
report_clocks -skew
```

	Latency	**Setup Uncertainty**
clk		
clk_alu		

- ✓ Confirm that there are four RAMs in this design (by convention all RAM instance names in this design are prefixed with "rams.")

```
get_cells -hier *rams.*
```

- ✓ Generate a timing report for paths to the RAMs.

```
report_timing -to */rams.*
```

- ✓ Fill in the first column using the timing report:
  - **Note:** A path margin of 2.0 ns was applied to the RAM endpoint. This was done solely for this lab to artificially create a large timing violation on the RAMs.

	Task 1	Task 2		Task 4
-to */rams.* or clk_alu group?	RAM	RAM	clk_alu	clk_alu
Launch clock network delay				
Data arrival time				
Capture clock network delay				
Data required time				
Slack				

- **Question 1.** Why can this violation not be fixed through traditional datapath logic optimization techniques?

  …………………………………………………………………………………

  …………………………………………………………………………………

  …………………………………………………………………………………

✓ Confirm that CCD optimization is enabled, and determine maximum postponing and preponing limits including macro skewing.

```
report_app_options compile.flow.enable_ccd
report_app_options ccd.*pone
report_app_options compile.flow.*skew_mac*
```

- **Question 2.** Without changing the default postponing and preponing limits, why is it unlikely that CCD optimization can fix the WNS violation*

  …………………………………………………………………………………

  …………………………………………………………………………………

  …..………………………………………………………………………………

- **Question 3**. Why does increasing the maximum "global" postponing and preponing limits not guarantee successful RAM timing?

  …………………………………………………………………………………

  …………………………………………………………………………………

  …………………………………………………………………………………

✓ Instead of increasing the maximum useful skew limits globally, we will next enable useful skew for macros, which focuses on improving the timing of RAM banks, at the possible expense of non-RAM timing paths.

✓ Generate a timing report for paths from the RAMs:

```
report_timing -from */rams.*
```

- **Question 4**. Based on your timing analysis, which "side" of the RAMs needs to be improved?

  …………………………………………………………………………………

  …………………………………………………………………………………

- **Question 5**. What is the default macro skewing effort level?

  ………………………………………………………………………………………

  ………………………………………………………………………………………

  ………………………………………………………………………………………

Task 2: Set up CCD and run compiIe_fusion

✓ Apply the commands to enable useful skew for macros and increase the macro skewing effort to *high*

```
set_skew_macros -bank_name reg_file_rams -macros /
[get_cells */rams*] -improve_side input
set_app_options -name compile.flow.skew_macros_max_postpone -value 0.9ns
set_app_options -name compile.flow.skew_macros_effort -value high
```

✓ In Fusion Compiler, CCD is on-by-default. therefore CTS setup should be performed. as discussed in the lecture. To do this. The review then sources the CTS setup file *scripts/cts_setup.tcl*

✓ Execute only the **initial_opto** stage of compile_fusion

```
source scripts/cts_setup.tcl
compile_fusion -from initial_opto -to initial_opto
```

✓ Generate another timing report for paths to the RAMs

```
report_timing -to */rams.*
```

✓ Return to page 30 and fill in the second column. Compare to the Task 1 results (note that the timing path may not be identical to Task 1. you might be looking at different RAM bits.)

✓ Generate a constraints report to get an overview of all tinting violations in the design:

```
report_constraints -all_violators -max_delay
```

✓ Based on the above report. analyze timing for non-RAM paths in the **clk_alu** clock domain:

```
report_timing -groups clk_alu
```

✓ Return to page 30 and fill in the third column. Compare to the Task 1 results

- **Question 6:** What adjustment(s) has CCD performed on that path? (Review Task 1. step 4 results)

  ………………………………………………………………………………………

  ………………………………………………………………………………………

  ………………………………………………………………………………………

Task 3: Analyze for potential CCD improvement

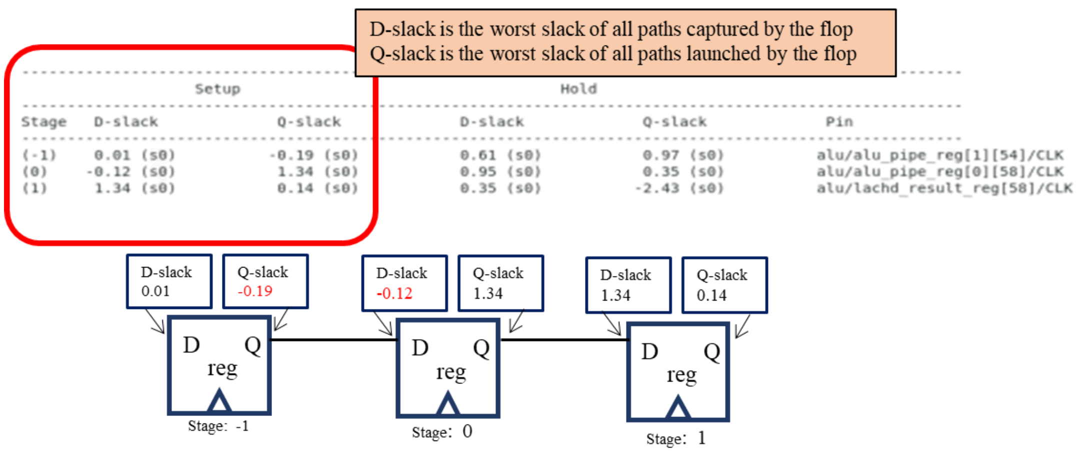
✓ To understand if it is possible to see more improvement from CCD. you may have to analyze the register stages before and after the critical register, in our case, alu/alu_pipe_reg[0] [58].

✓ You could use the command report_timing command to analyze the registers before and after, but **report_ccd_timing** makes this much easier. The **report_ccd_timing** command prints **D-** and **Q-** slacks of critical endpoints and provides options to analyze the timing paths of the previous and following stages

✓ Analyze the stages immediately surrounding the critical register:

```
report_ccd_timing –pins alu/alu_pipe_reg [0] [58]/CLK
```

✓ You should see the following output (we're showing only the setup column)



D-slack is the worst slack of all paths captured by the flop
Q-slack is the worst slack of all paths launched by the flop

```
-----------------------------------------------------------------------------------------------------
                  Setup                                              Hold
-----------------------------------------------------------------------------------------------------
Stage   D-slack               Q-slack              D-slack              Q-slack              Pin
-----------------------------------------------------------------------------------------------------
(-1)     0.01 (s0)           -0.19 (s0)            0.61 (s0)            0.97 (s0)            alu/alu_pipe_reg[1][54]/CLK
(0)     -0.12 (s0)            1.34 (s0)            0.95 (s0)            0.35 (s0)            alu/alu_pipe_reg[0][58]/CLK
(1)      1.34 (s0)            0.14 (s0)            0.35 (s0)           -2.43 (s0)            alu/lachd_result_reg[58]/CLK
```

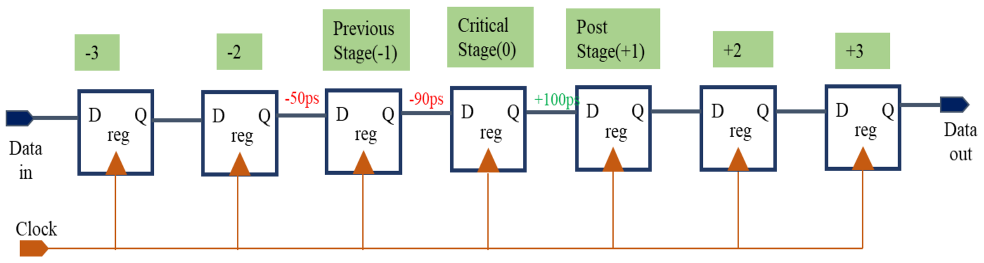
- ✓ Focus on the set up0 column. There you can see that the D-slack timing violation of the middle stage (0) is -0.15 and the previous stage has a negative Q-slack of -0.154. The next stage (1) has a positive D slack of 1.408. This means that the stage (0) register could be postponed without impacting stage (1).
- ✓ If you look back at the tinting report, you will see that the stage (0) register has already been postponed by around 0.2ns. To postpone this register further, you will have to change the max postpone default.
  - **Note:** In our specific example the Q-slack and D-slack numbers between stages (0) and (1) match each other. This is not always the case. That's because the Q-slack represents the worst timing path to all capture endpoints launched by that register, while the D-slack represents the worst tinting path from all launch start points to that capture register.
- ✓ Use the following command to have a different view of the stages before and after the register in question:

```
report_multistage_timing
```

- ✓ Examine a different path, this time using the chain type:

```
report_ccd_timing –type chain –pins\stack_top/stack_mem1/stack_mem_reg[0] [2]/CLK
```

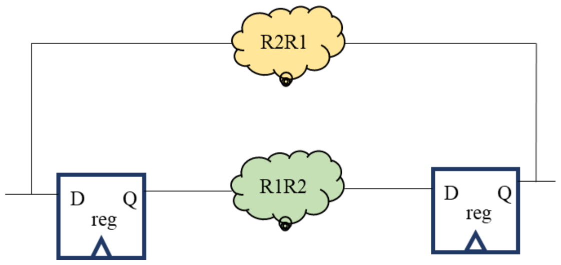
- ✓ This will show the entire chain of critical registers involved in this path.
- ✓ Our chain starts at an input port and ends at a register. Here's an example chain.



- ✓ In the above example, the two stages with a -50ps violation could be postponed to take advantage of the +100ps slack.
- ✓ In the chain reported, you will see three registers tagged with L1. This means that these 3 registers form a loop, meaning that the last register has a data path looping back to the first L1 register.
- ✓ You can confirm this by running the following timing report:

```
        report_timing \
            -from stack_top/stack_mem1/popdataout_reg[2]\
            -to prgrm_cnt_top/prgrm_cnt/point_reg[2]
```

- ✓ CCD can be problematic with loops. Here's another example.



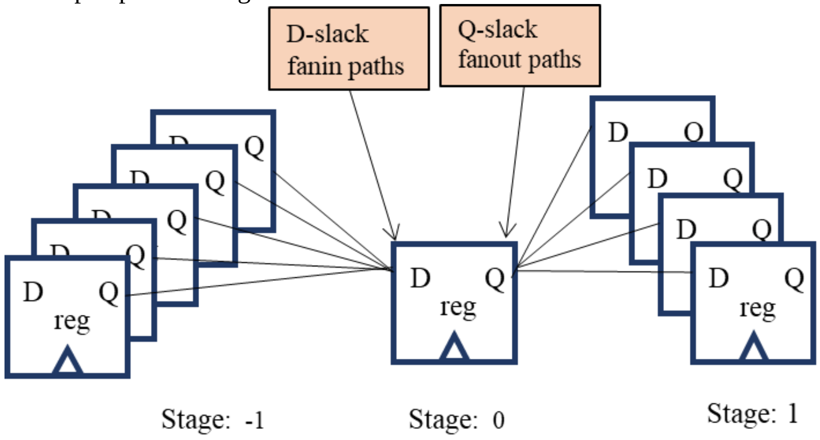
- ✓ Assume that each of the paths above has a -50ps violation. To fix R1R2, reg2 is postponed by 50ps. This in turn will increase the violation of R2R1 to -100ps to fix R2R1, reg1 would have to be postponed by 100ps. which in turn worsens R1R2 even more. You get the idea…
- ✓ Finally, you can examine the fanout and fanin to the critical register (the one we focused on at the beginning of this task). This can help you understand the scope of the pre-and postponing that needs to occur.
- ✓ Run the following commands.

```
            report_ccd_timing
                        -pins alu/alu_pipe_reg [0][58]/CLK –type fanin\
            report_ccd_timing
            report_ccd_timing
                        -pins alu/alu_pipe_reg[0] [58]/CLK –type fanout
```

- ✓ By default, the fanin/fanout reports will only list up to 5 timing paths. To see more use the –max_paths switch.
- ✓ The fanin report shows 5 paths, which means there could be more. Confirm using:

```
          report_ccd_timing
            -pins alu/alu_pipe_reg [0][58]/CLK\
            -type fanin –max_paths 40
```

- ✓ You will find 32 paths, all of them with a negative slack.
- ✓ Looking at the fanout paths. You will only find one. Since this one path has a positive slack it should be no problem to postpone the register.Here's a schematic to demonstrate fanin/fanout cones



- ✓ Consider the positive fanout slack and answer the following.
  - **Question 7:** How many fanin paths can potentially be fixed only by postponing the critical register?

    ………………………………………………………………………………………

    ………………………………………………………………………………………

    ………………………………………………………………………………………

Task 4: Continue with *compile_fusion*

- ✓ As you have seen from the previous analysis for the clock **clk_alu**, the maximum postpone limit was reached. You will next increase the maximum postpone limit, then run the last two stages of**compile_fusion**, **final_place,** and **final_opto,** Remember that two more useful skew adjustments are performed during **final_opto**
- ✓ Set the maximum to postpone option to 400ps, instead of the default 200ps.

```
set_app_options –name ccd.max_postpone –value 0.4
```

- ✓ Run **compile_fusion** from the **final_place** to the **final_opto** stage.

```
compile_fusion –from final_place –to final_opto
```

- ✓ Repeat timing analysis of the RAM and the non-RAM paths

```
report_timing –to */rams.*
report_timing –groups clk_alu
```

- ✓ Return to page 30 and fill in the fourth and fifth columns, if you like
- ✓ You should find that you have no minuscule tinting violations.
- ✓ Exit out of Fusion Compiler (Discard All when prompted)

Lab 06 - Managing Integrated Clock Gating (ICGs)

Objective : *To get familiarized with the insertion and optimization of ICGs during compile*

Working Directory : /home/<user_name>/PD_LABS/FC/lab06/

Project Name : ORCA_TOP.dlib

Learning outcomes:

- ✓ Generate and analyze a clock gating report
- ✓ Apply clock gate transformations
- ✓ Determine the latencies the tool applies to ICGs
- ✓ Analyze timing to ICG to enable pins
- ✓ Determine tool-inserted and pre-existing ICGs

Task 1: Initial Map

- ✓ Invoke Fusion Compiler' from the lab directory:

```
UNIX% cd lab_6
UNIX% fc_shell -gui
```

- ✓ From the script editor tab select open (folder) and, if needed, use the Parent Directory up-arrow to open the **1ab06** directory. then open **run.tcl**.This file contains all the commands that you will be executing in this lab. Select the commands from this file. Check the Selected box and press Play instead of typing them yourself, to save time and avoid typing errors.
- ✓ There is an existing block called **risc_core/setup** which has gone through all the design setup steps. No ICG-related commands or application options have been applied to this block. Copy the block to **risc_core/icg:**

```
open_lib risc_core.dlib
copy_block -from risc_core/setup -to risc_core/icg
current_block risc_core/icg
```

- ✓ Generate a clock gating report and fill in the pre-compile column of the table:

```
alias rcg report_clock_gating
rcg
```

	Pre Compile	Initial_map	Logic_opto	Initial_opto	Final_opto
Number of Clock gating elements					
Number of Tool-Inserted Clock gates					
Number of Pre-Existing Clock gates					
Number of Gated registers					
Number of Tool-Inserted Gated regs					
Number of Pre-Existing Gated regs					
Number of Ungated registers					
Total Number of registers					

Max Number of Clock Gate Levels					
Number of Multi Level Clock Gates					

✓ Some key observations that you can make from this report:

- There are no tool-inserted CGs. but there are 6 pre-existing CGs, that is one level deep
- Many ungated registers can potentially be gated by the tool
- **Question 1.** What are the default minimum and maximum register bitwidth/fanoutlimits for tool inserted ICGs? (**Hint**: run the command without any options or view the man page **for set_clock_gating_options**)

  ……………………………………………………………………………………

  ……………………………………………………………………………………

  ……………………………………………………………………………………

✓ Check whether ideal clock latencies were applied using: `report_clcok –skew`

- **Question 2:** What latencies do the clocks clk and clk_alu have?
  ……………………………………………………………………………………
  ……………………………………………………………………………………
  ……………………………………………………………………………………

✓ It is advisable to apply key CTS setup constraints since ICG insertion relies on library cell selection, and CCD and ICG optimization run CTS operations. Have a look at the life file script/cts_setup.tcl and source it:

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

✓ Find out what ICGs are enabled for insertion. Only ICGs with the library cell purpose "cts" will be inserted by the tool. Pre-existing ICGs are not affected, but may be sized to cts-purpose ICGs:

```
get_lib_cells */CG* -filter valid_purposes=~*cts*
```

✓ Get the names of the pre-existing ICGs : `get_clock_gates`

✓ You should see the following six pre-existing ICGs

```
alu/alu_cg
data_path/data_cg
reg_file/rama_cg
reg_file/ramb_cg
reg_file/ramc_cg
reg_file/ramd_cg
```

✓ Let's focus on the **RAM ICGs**.

✓ Get a list of all the pins of one of the clock gates.

```
get_pins -of_objects reg_file/rama_cg
```

✓ You should see the pins that matter to us, namely the ICG clock input and output pins, and the clock enables pin: CLK, GCLK, and EN.

✓ Find the clock pins that are driven by one of the ICGs:

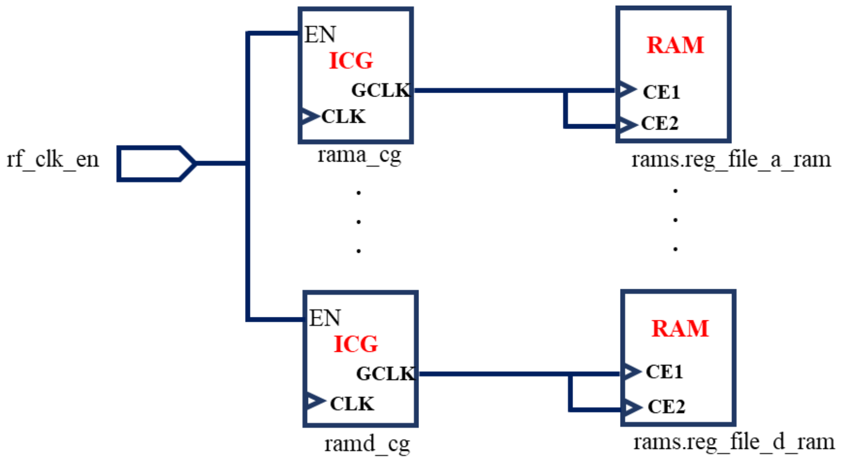
```
all_connected [get_nets -of_objects reg_file/rama_cg/GCLK]
```

✓ You should see that the ICG connects to clock pins CE1 and CE2 of a RAM called rams.reg_file_a_ram. You can check ramb_cg, ramc_cg, and ramd_cg as well if you like. All the ram*_cg ICGs connect to the corresponding rams.reg_file_*_ram RAMs.

✓ Find out what controls the ICGs:

```
all_fanin -flat -to reg_file/rama_cg/EN
```

✓ The ICG is controlled by the input port **rf_clk_en**. If you check the other RAM-ICGs. You will find that all of them are controlled by the same enable.



✓ Prevent Fusion Compiler from merging the ICGs of **a_** and **b_rams** with other ICGs (c_ and d_rams are allowed to be merged - this is being done for illustration purposes later on):

```
set_clock_gate_transformations [get_cells "reg_file/rama_cg
reg_file/ramb_cg"] true -merge_equivalents false
```

✓ Run the first stage of compile, **initial_map:**

```
_run_compile_stage initial_map
```

- **Note:_run_compile_stage** is a Tcl procedure that runs an individual compile stage and creates a log file for the executed stage. It is a little simpler to enter compared to compile_fusion –from <stage> -to <stage>. This is not a native command to Fusion Compiler. This procedure was created specifically for this workshop.

✓ Generate another clock gating report and capture the new data in the initial map column of the table on page 35: `recg`

- **Question 3:**How many ICGs were inserted during initial_map?
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………
- **Question 4:** Why are there ungated registers?
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

✓ Using the same **rcg -ungated** output:

- **Question 5:** How many registers are only gated by the pre-instantiated clock gates?
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

✓ Have look at the log file compile_fusion.initial_map.log and answer the next question.

- **Question 6:** Why did the number of registers go down from the pre-compile register count?
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

✓ Verify your answer with this report

```
report_transformed_registers
```

- **Note:** The report only lists registers that were transferred during compilation. Executing the command before compiling returns an empty list. In other words, this cannot be used to identify what is going to happen during compile

✓ Save the block: `save_block`

Task 2: Logic Opto

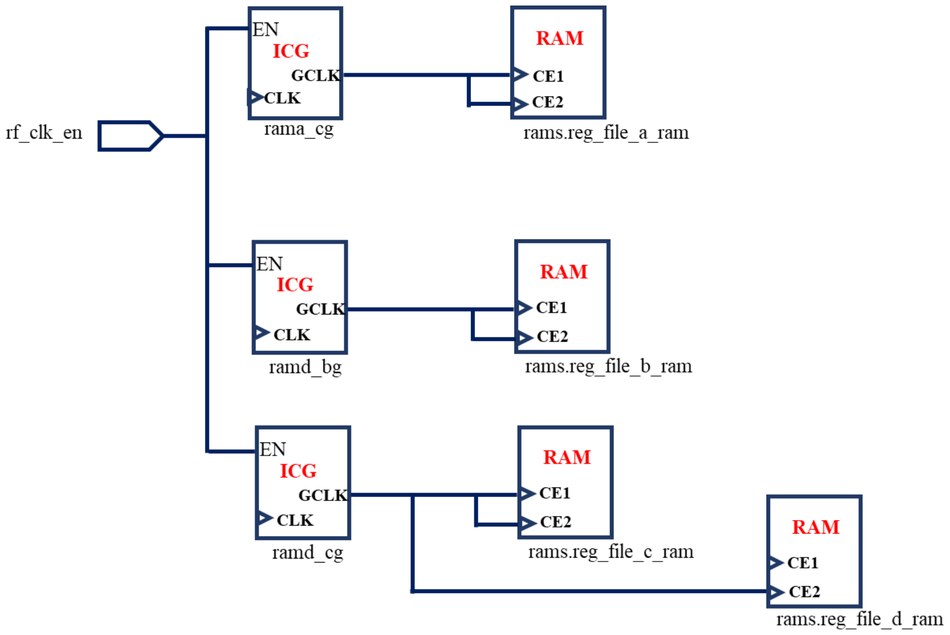
- ✓ Run the second stage of compile, logic_opto:

```
_run_compile_stage logic_opto
```

- ✓ Generate a clock gating report and capture the new data in the logic opto column of the table on page-35:rcg
  - **Note:** The number of ICGs was reduced from 24 to 21, as a result of two tool-inserted ICGs being removed (18→ 16) and one pre-existing ICG being removed (6 5).
- ✓ Run the following report and you will find that a couple more registers are no longer gated due to min-bit width:rcg -ungated
- ✓ Clock gates are inserted very early' during **initial_map**. The reduction in ICGs at the beginning of **logic_opto** is due to the register merging that occurs late in **initial_map**. (you can review the logfile **compile_fusion.initial_map.log** confirm).
- ✓ Next. investigate the rem → oval of one of the pre-existing ICGs.
- ✓ Run the following commands to check whether the RAM ICGs are still connected to their corresponding RAMs:

```
all_connected [get_nets -of_objects reg_file/rama_cg/GCLK]
all_connected [get_nets -of_objects reg_file/ramb_cg/GCLK]
all_connected [get_nets -of_objects reg_file/ramc_cg/GCLK]
all_connected [get_nets -of_objects reg_file/ramd_cg/GCLK]
```

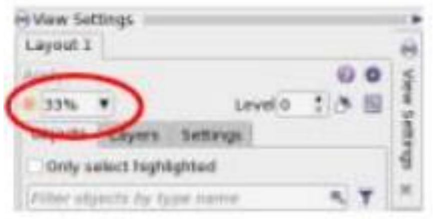
- ✓ You should find that ramc_cg now connects to the CE pins of c_ram as well as d_ram, and that ramd_cg no longer exists. Because of the set_clcok_gate_transformations rules you set up in Task1, rama_cg and ramb_cg were not merged, but ramc_cg and ramd_cg were, because they,

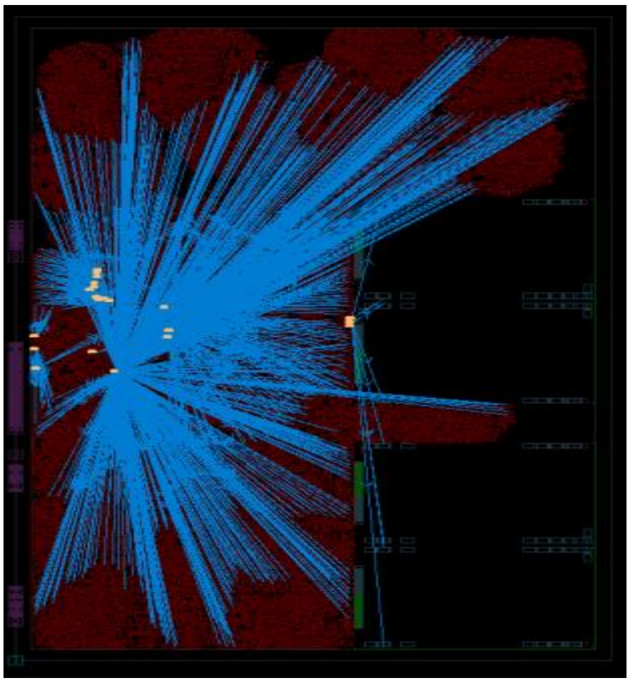


- ✓ Next, we would like to analyze the placement of the ICGs in relation to their register sinks.
- ✓ Since there is no command to analyze ICG placement, we created a procedure called h1_icgs which highlights the ICGs in the GUI in orange and highlights the nets connected to each ICG's output pin. The procedure will also work if buffers exist between the ICGs and the registers, which will be useful for analysis after trial CTS. Have a look at the procedure, then run it.

```
info body hl_icgs
hl_icgs
```

- ✓ In the GUI, use the View Settings panel on the right to reduce the brightness of the non-highlighted objects to 33% for better visibility.





- ✓ The layout should look like this.

```
gui_set_setting -window [gui_get_current_window -types Layout -mru] -
        setting brightness -value 33%
```

- ✓ Report timing for paths ending at ICG **<u>enable</u>** pins:

```
rt -to [get_clock_gate_pins -type enable]
```

  - **Note:** rt is an alias for report_timing –cap –tran –sign 3, defined in ../ref/tools/procs.tcl
- ✓ You will find that the critical ICG enable setup timing is the manually inserted clock gate,alu_icg.
- ✓ Capture the following:
  - Pre-existing ICG enables setup timing slack:……………
  - Pre-existing ICG clock network delay: ……..…….
  - **Note:** The placement at this point is an initial rough placement, and timing optimization has not occurred on the placed netlist yet.
- ✓ Execute the following:`write_script`
- ✓ Look at the file wscript/scenario+func::ss_125c.tcl. You should be able to find the tool-estimated clock gate latency for alu/alu_cg/CLK.
  - **Note:** Notice that the latency "offset" does not match the ICG's clock network delay reordered in the previous step. That's because the offset is applied relative to any user-defined ideal network latency that may exist on the ICG's clock. At the top of the output script, you see that there is a 1.0ns user-defined clock latency on **clk_alu.**
- ✓ Run the following two commands to show the enable timing to the second level ICG after the user-inserted alu_cg:

```
filter_collection [all_fanout -flat -only_cells
-from alu/alu_cg/GCLK] "ref_name=~*CG*"
rt -to [get_pins alu/clock_gate_lachd_result_reg/EN]
```

- ✓ You should find that the enable of that ICG meets timing easily. Also, notice that the network latency on this second-level ICG's CLK pin is larger Than the first-level **alu_cg** cell.
- ✓ You can find out the names of the pre-existing and tool-inserted ICGs at any time by using:

```
get_clock_gates -origin pre_existing
get_clock_gates -origin tool_inserted
```

- ✓ Generate timing report for the most critical tool-inserted ICG enable pin:

```
rt -to [get_clock_gate_pins -type enable -of_objects [get_clock_gates -origin tool_inserted]]
```

  - Tool-inserted ICG enables setup timing slack:…………….
  - Tool-inserted clock network latency: ………………....

- ✓ You should find that there are no enable timing violations to any tool-inserted ICGs.
- ✓ Save the block: `save_block`

Task 3: Initial Place to Initial Opto

- ✓ Run the initial_place, initial_drc as well as initial_opto compile stages:

```
compile_fusion -from initial_place -to initial_opto
```

- ✓ **initial_place** and **initial_drc** should not make changes to the ICGs, except for the placement. During the **initial_opto** stage, ICGs can be optimized to meet the constraints (e.g. fan out) that you have set earlier.
- ✓ Generate a clock gating report and capture the new data in the **initial_opto** column of the table on page 35: `rcg`
- ✓ You should find that no changes to the number of ICGs have occurred, but the register count and the number of gated registers increased slightly. The increase is due to a few registers that were "un”-merged. They were merged earlier because they were equivalent but unmerged now because they became timing critical due to their large fanout. You can run a timing report from one of the replicated registers

```
rt -nets -from *_rep*
```

- ✓ The unmerged registers will show up as -replicated in this report:

```
report_transformed_registers
```

- ✓ Have a look at the placement of ICGs and compare them to the results in Task 2: `hl_icgs`
- ✓ Generate max-delay constraints report summarizing all setup timing violations:

```
rc -max_delay -sig 3
```

  - **Note:** rc is an alias for report_constrints –all_violators
- ✓ You should see a very small setup timing violation to the enable pin of a pre-existing ICGs .(and possibly a negligible violation to a RAM).
- ✓ Capture the timing result:
- ✓ Pre-existing ICG enables WNS: ……………
- ✓ Compare this result to your answer in Task 2.
- ✓ Generate timing reports to the tool-inserted and pre-existing ICGs. and capture the latencies to their respective clock pins

```
rt -to clock_gate*
rt
```

  - Tool-inserted ICG clock network delay: ………………
  - Pre-existing ICG clock network delay: ……………………
- ✓ You should see that the clock pin latencies have changed due to the additional placement and optimization runs during **initial_place** and **initial_opto**
- ✓ Save the block: `save_block`

Lab 07 - Introduction to Floorplanning

Objective : *To get familiarized with the basic block-level floorplanning operations in Fusion Compiler*

Working Directory : /home/<user_name>/PD_LABS/FC/lab07/

Project Name : ORCA_TOP.dlib

Learning outcomes:

- ✓ Define a rectilinear block shape
- ✓ Place voltage areas inside the black
- ✓ 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

Task 1: Invoke Fusion Compiler and load the ORCA_TOP Block

- ✓ Invoke Fusion Compiler' from the lab_7 directory

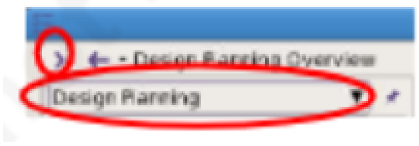
```
UNIX% cd lab07
UNIX% fc_shell -gui
```

- ✓ **Invoke run.tcl file** in **S**cript Editor to simply highlight/select lines you want to execute, and then click on the Run (play) icon present in the script editor
- ✓ Open the block which has been initialized with Verilog, UPF, and timing constraints, ready to be floor planned. Use the GUI (Open an existing block), or select/run the following command:

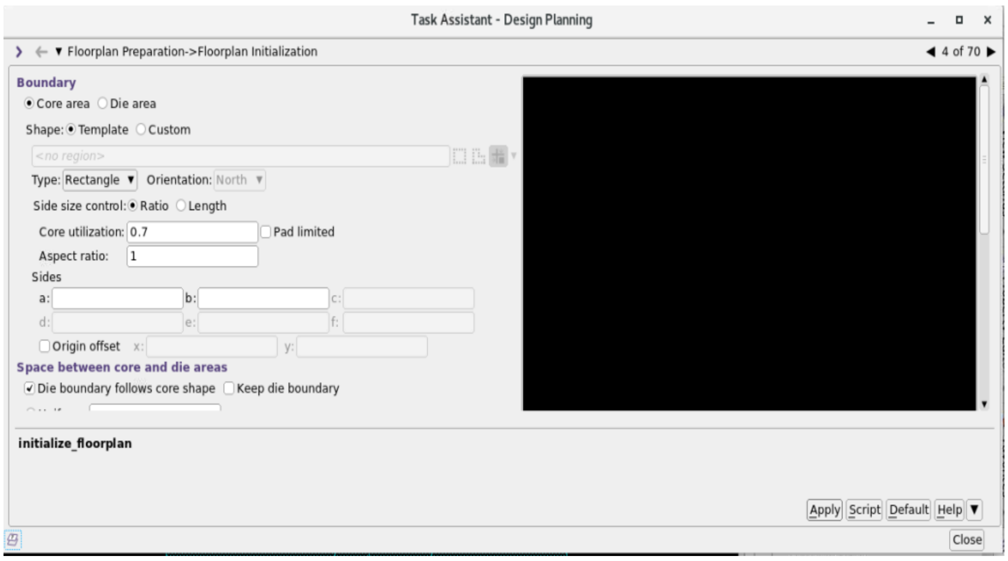
```
open block ORCA TOP.dlib: ORCA TOP/floorplan
```

Task 2: Initial Floorplanning using the Task Assistant

- ✓ 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**.



- ✓ Select **Floorplan Preparation → Floorplan Initialization**. This opens the *Floorplan Initialization* dialog as show below


Task Assistant - Design Planning

Floorplan Preparation->Floorplan Initialization 4 of 70

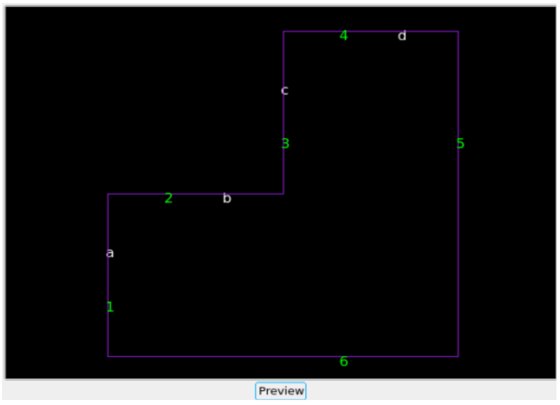
Boundary
[✓ Core area] Die area
Shape: [✓ Template] Custom
<no region>
Type: Rectangle Orientation: North
Side size control: [✓ Ratio] Length
Core utilization: 0.7 Pad limited
Aspect ratio: 1
Sides
a: b: c:
d: e: f:
Origin offset x: y:
Space between core and die areas
[✓ Die boundary follows core shape] Keep die boundary

initialize_floorplan

✓ Use the boundary Type and Orientation pull-down fields to create the block L-shape shown here (click on Preview under the black Display Window in the dialog to double-check your setup):

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

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

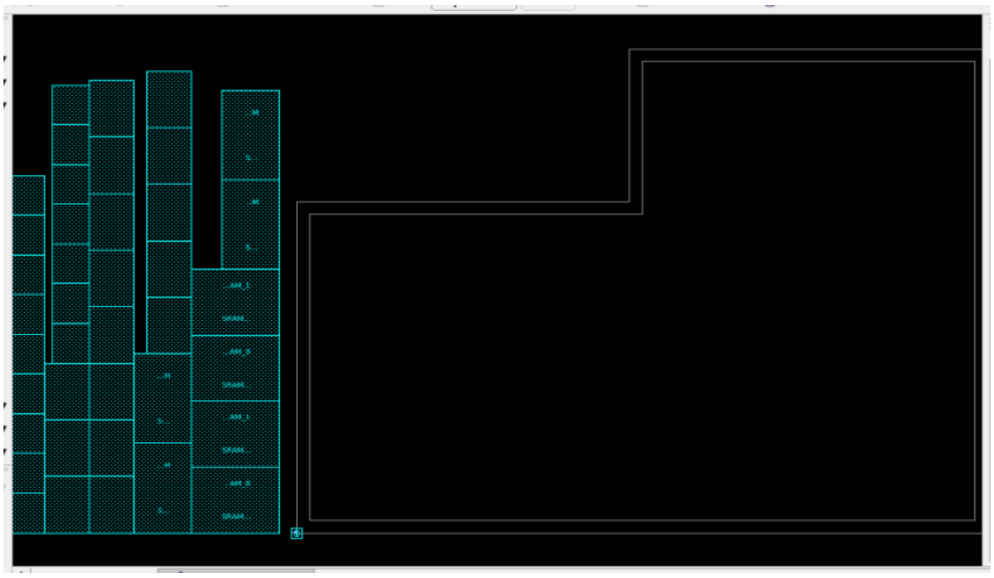


- **Hint**: Use whole numbers only. The core shape should be set to "L-shape", and Orientation should be set to "West".Set the 4 parameters as follows: a=2, b=2, c=1, d=2

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

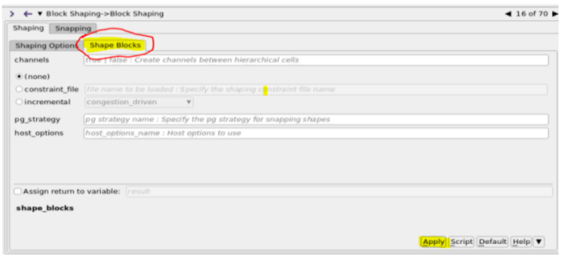
✓ Click on Apply to generate the initial floorplan and then Close the Task

✓ You should now see something like this in the Block Layout Window

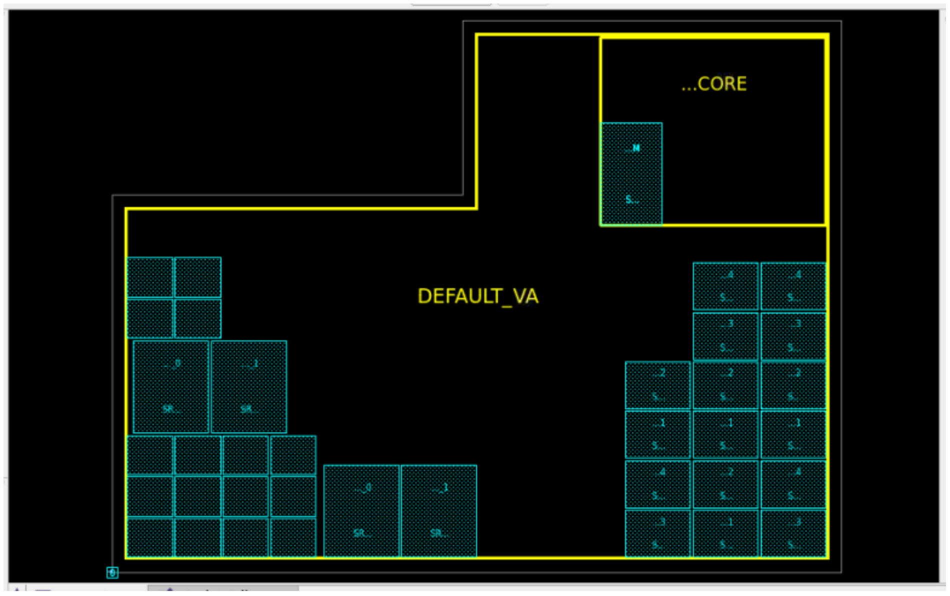


Task 3: Block Shaping

- ✓ From the *Tasks palette*, select **Block Shaping** ➔ **Block Shaping**: This is used to automatically create (shape) voltage areas.
- ✓ Select **Shape Blocks** and click**Apply**.
- ✓ After ***shape_blocks*** completes, bring the layout view to the foreground.


Block Shaping->Block Shaping
16 of 70
Shaping
Snapping
Shaping Options
Shape Blocks
channels
true | false : Create channels between hierarchical cells
(none)
constraint_file
file name to be loaded : Specify the shaping constraint file name
incremental
congestion_driven
pg_strategy
pg strategy name : Specify the pg strategy for snapping shapes
host_options
host_options_name : Host options to use
Assign return to variable:
result
shape_blocks
Apply
Script
Default
Help

- ✓ Enable visibility of voltage areas by selecting Voltage Area from the **ViewSettings** ➔ **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 comer, 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**: Four macros inside that voltage area are stacked 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.

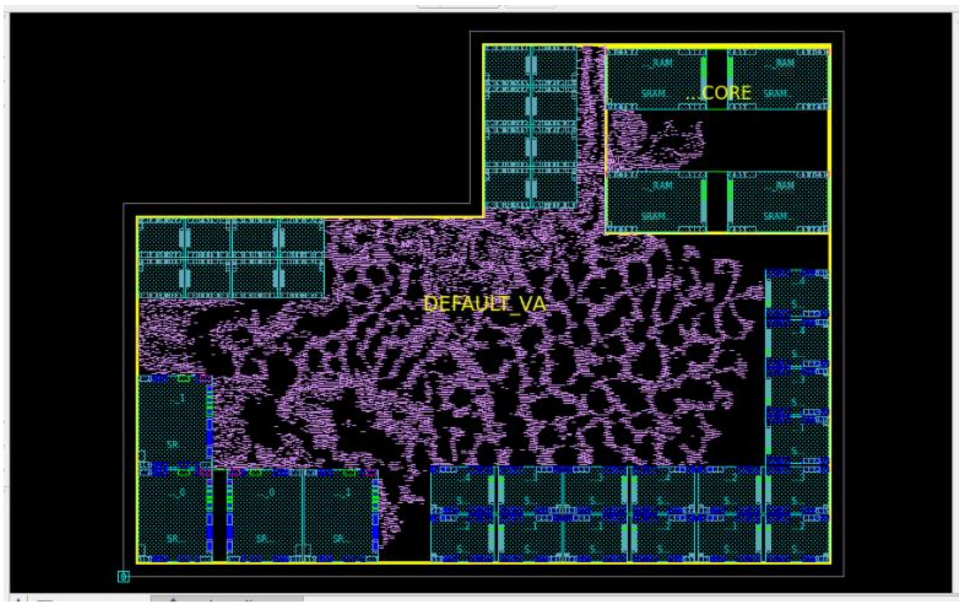


Task 4: Macro and Standard Cell Placement

- ✓ From the Tasks palette, select Cell Placement ➔ Cell Placement
  - **Note:** There is a **Cell Placement** ➔ **Hard MacroConstraints** 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**.
- ✓ If you want to try out free-form macro placement, run the following command to enable it:

```
set_app_options -name plan.macro style -value freeform
```

- ✓ Run placement by clicking on **Apply**.
- ✓ 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 common pins face each other. This is done to minimize the overall number of channels that require routing and possibly buffering between macros.

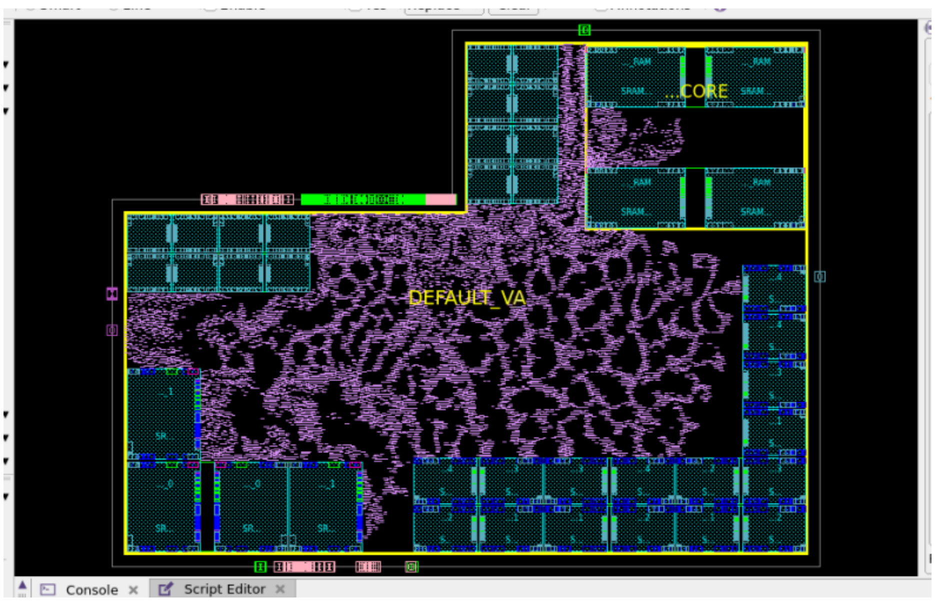


Task 5: Place the Block Ports

- ✓ You could also use the *TaskAssistant* to place the pins of the block, but this time just select/run the corresponding commands from run. tcl:

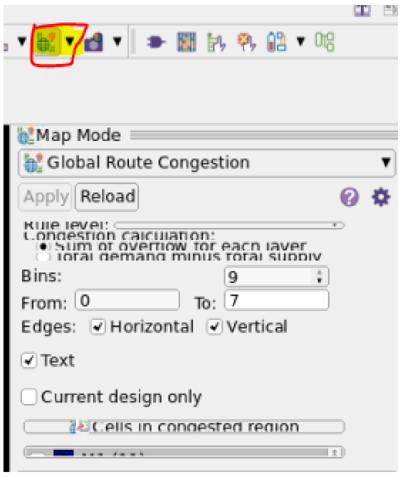
```
set_block_pin_constraints -self  -allowed_layers {M3 M4 M5 M6}
place_pins -self
```

- ✓ Have a look at the layout, you should see that all the block's ports, the logical representations of the physical pins, have been placed.
- ✓ 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).
- ✓ Search for the ***clk** ports/terminals and zoom into their location. See**run.tcl** for hints.
- ✓ 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. Afterward, 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.
- ✓ 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.



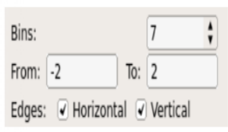
Task 6: Congestion Map

- ✓ 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.
- ✓ Bring up the congestion map by selecting the "*Global Route Congestion*" entry from the *GUI maps* pull-down menu.



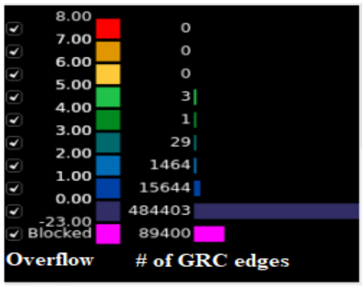
Map Mode
Global Route Congestion
Apply Reload
Rule level:
Congestion calculation:
● Sum of overflow for each layer
○ Total demand minus total supply
Bins: 9
From: 0 To: 7
Edges: ✓ Horizontal ✓ Vertical
✓ Text
☐ Current design only
Cells in congested region

- ✓ Select "**Reload**", then confirm the dialog that appears by pressing **OK**.
- ✓ 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.
- ✓ To make the display a little more interesting, try the following steps.
- ✓ Change the **Bins/From/To** settings as shown in the following screenshot:



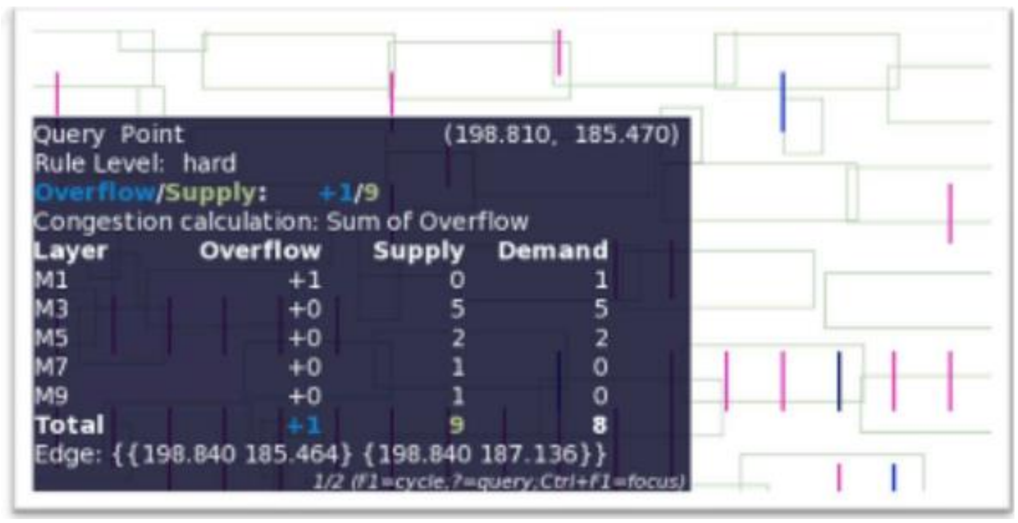
Bins: 7
From: -2 To: 2
Edges: ✓ Horizontal ✓ Vertical

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



	Overflow		# of GRC edges
✓	8.00	🟥	0
✓	7.00	🟧	0
✓	6.00	🟨	0
✓	5.00	🟩	3
✓	4.00	🟩	1
✓	3.00	🟩	29
✓	2.00	🟦	1464
✓	1.00	🟦	15644
✓	0.00	🟦	484403
✓	-23.00 Blocked	🟪	89400

- ✓ 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 overflows.
- ✓ 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:



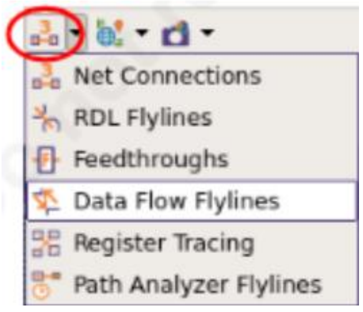
```
Query Point                              (198.810, 185.470)
Rule Level: hard
Overflow/Supply:     +1/9
Congestion calculation: Sum of Overflow
Layer          Overflow     Supply    Demand
M1                   +1          0         1
M3                   +0          5         5
M5                   +0          2         2
M7                   +0          1         0
M9                   +0          1         0
Total                +1          9         8
Edge: {{198.840 185.464} {198.840 187.136}}
                   1/2 (F1=cycle,?=query,Ctrl+F1=focus)
```

✓ Close the congestion map, either by clicking on the Draw Global RouteCongestion ... icon in the top banner or by closing the palette.

Task 7: Analyze Macro Placement using Data Flow Flylines (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.

✓ Bring up *Data Flow Lines* by selecting the appropriate entry from the connectivity pull-down, located to the left of the *maps* pull-down.



✓ 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.

✓ Once this completes, select **Macros and Ports** for items to Include in the DFF analysis and click on Apply (highlighted in blue once you make changes) to accept the changes.

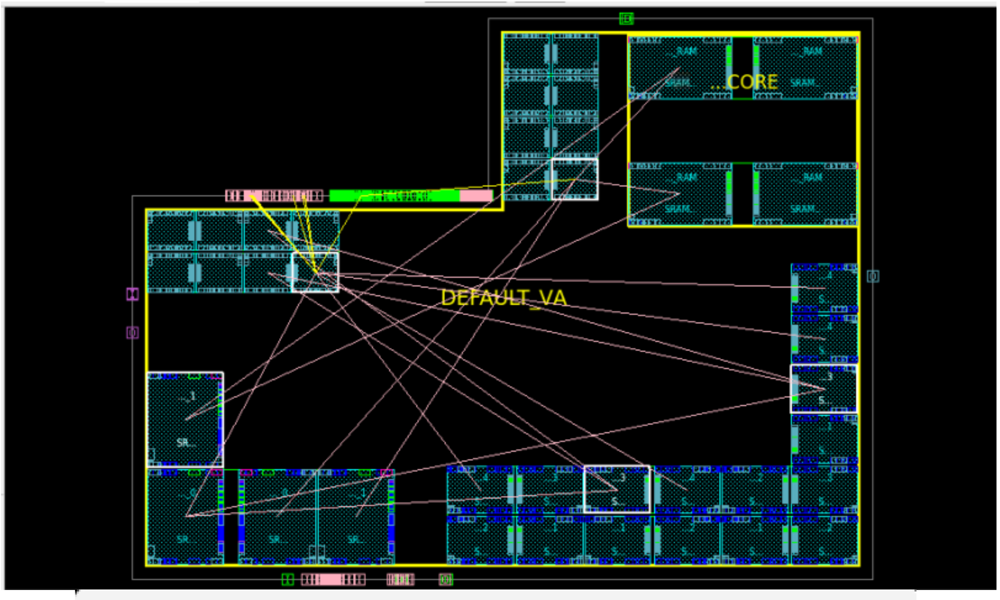
✓ Now select one macro to see its connections to other macros and ports. **Limit the tracing** by checking the "**Number of registers**" or "**Number of gates**" and changing the **Min/Max** numbers. Using this method, you can quickly figure out whether objects are connected directly, through several gate levels, or several register levels.

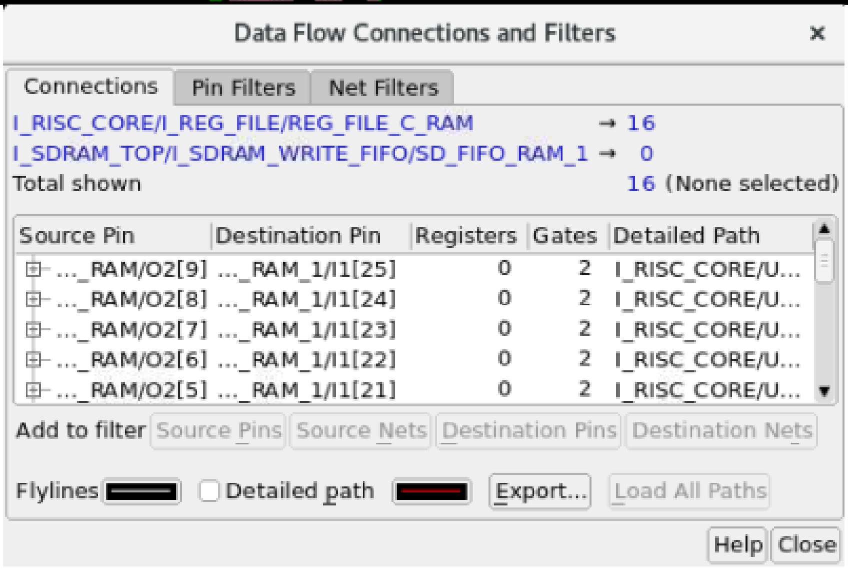
✓ You can select several macros at the same time using the *Control* key. If you click on a flyline, you will get detailed information about the connection(s).

✓ 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.

✓ Close the DFF palette.




Data Flow Connections and Filters

Connections | Pin Filters | Net Filters

I_RISC_CORE/I_REG_FILE/REG_FILE_C_RAM → 16
I_SDRAM_TOP/I_SDRAM_WRITE_FIFO/SD_FIFO_RAM_1 → 0
Total shown 16 (None selected)

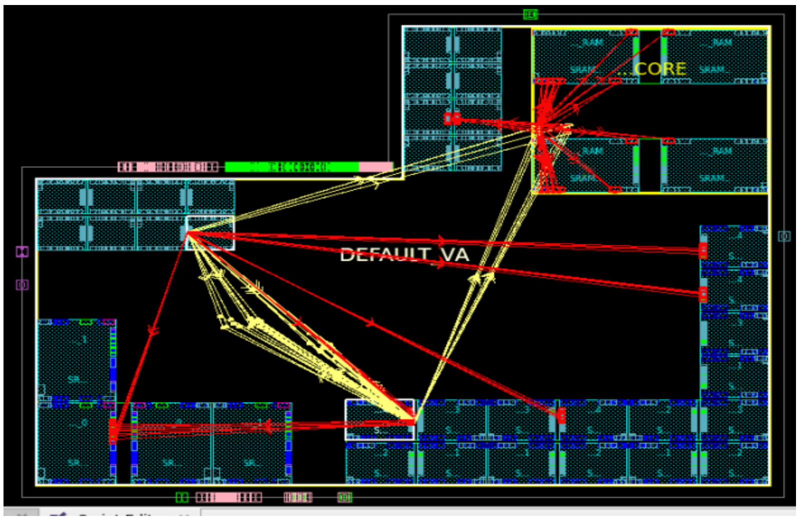
Source Pin	Destination Pin	Registers	Gates	Detailed Path
..._RAM/O2[9]	..._RAM_1/I1[25]	0	2	I_RISC_CORE/U...
..._RAM/O2[8]	..._RAM_1/I1[24]	0	2	I_RISC_CORE/U...
..._RAM/O2[7]	..._RAM_1/I1[23]	0	2	I_RISC_CORE/U...
..._RAM/O2[6]	..._RAM_1/I1[22]	0	2	I_RISC_CORE/U...
..._RAM/O2[5]	..._RAM_1/I1[21]	0	2	I_RISC_CORE/U...

Add to filter Source Pins Source Nets Destination Pins Destination Nets
Flylines ☐ Detailed path Export... Load All Paths
Help Close

Task 8: Register Tracing

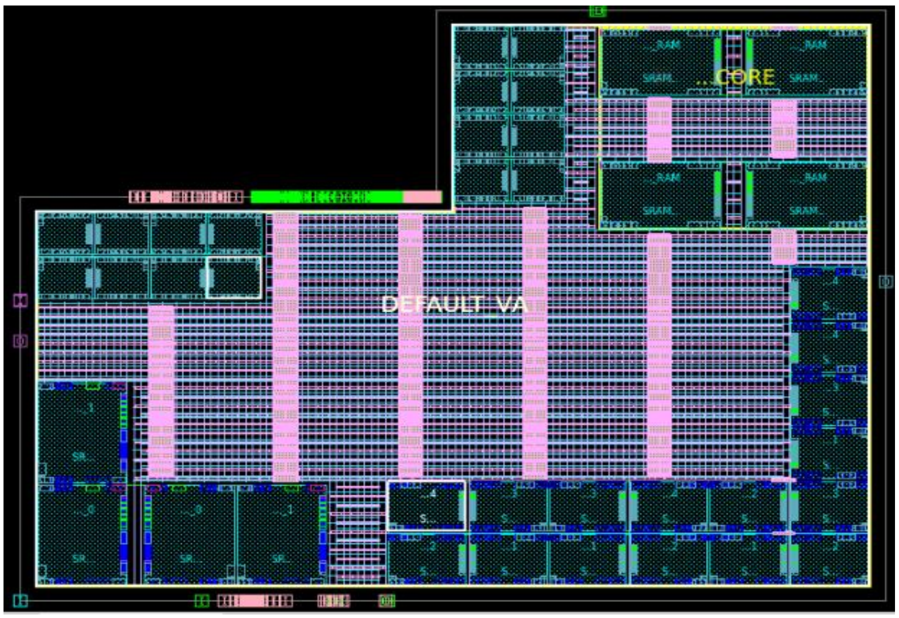
- ✓ Select *Register Tracing* from the *connectivity* pull-down, located right under the *Data Flow Fylines* entry.
- ✓ Select any macro. The registers connecting to that macro will be highlighted immediately, skipping any logic that may be in between. Select Show *flylines* to see the flylines between the macro and registers.
- ✓ To see the next level of registers, increase Max levels to 2 under the **LimitTracing** heading, and under **Show Levels**, check **Level 2**. If there is a second level of registers they will be highlighted in a different color.
- ✓ Under Highlight, you have the option to also display *Endpoints*: These are the endpoints from the last level of registers displayed. You can also display *Direct endpoints*, which are endpoints that connect directly (level 0) to the selected source, i.e. our macro.
- ✓ Close the Register Tracing palette.
- ✓ 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 a 🔒▾ or by entering:

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



Task 9: Power and Ground (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 placeholder for the final power mesh, to check congestion issues that a PG mesh may cause.
- ✓ From the *Tasks palette*, select **PG Planning** ➔ **PG Prototyping**
- ✓ 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.
- ✓ Specify **M8** and **M7** instead of the default vertical and horizontal layers, respectively, using the pull-down menus.
- ✓ Click on **Apply**. In a couple of seconds, you should see a basic PG mesh inserted, which does not route over hard macros.
- ✓ You can remove the PG mesh by clicking on **Remove PG Routes**.
- ✓ If you like, play around with the layer and percentage parameters to test different PG mesh configurations.
- ✓ Remove PG routes before re-applying with new parameters.
- ✓ In the next task, you will build the final mesh using a pre-written script.



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 Fusion Compiler using pattern-based power network synthesis(PPNS) and to demonstrate how few commands it takes to create a full multi-voltage PG mesh.
- ✓ 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_pa_*_pattern), followed by the strategies(set_pg_strategy). Once both are defined, the strategies are implemented using compile_pg.
- ✓ Source the script: `source -echo scripts/pns.tcl`
- ✓ Once the script has been completed, review the power mesh, the macro PG connections, and the standard call 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.
- ✓ Save and exit the tool:

```
        save_lib
         exit
```

Lab 08(a) - Setting up CTS

Objective : *To get familiarized with the setup steps for clock tree balancing, NDRs as well as timing/DRCs*

Working Directory : /home/<user_name>/PD_LABS/FC/lab08a/

Project Name : ORCA_TOP.dlib

Learning outcomes:

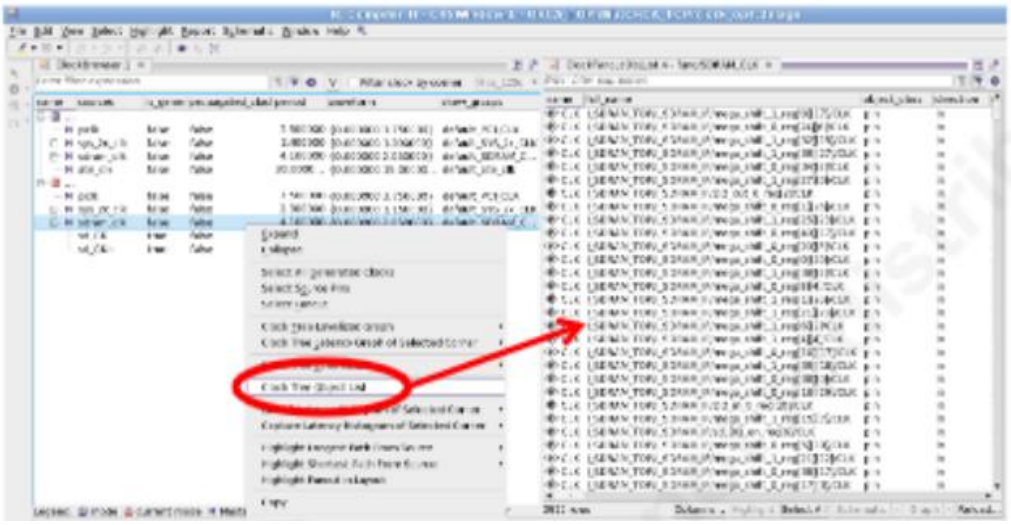
- ✓ Set up clock tree balancing
- ✓ Create and apply Non-Default routing rules
- ✓ Apply clock-related timing and DRC constraints

Task 1: Load the Design and Analyze the Clocks

- ✓ Change to the working directory for thelab08a, then load the starting design:

```
UNIX% cd lab08a
UNIX% fc shell -gui -f load.tcl
```

- ✓ The script will open a block that is ready for the upcoming tasks.
- ✓ Open the **run. tcl** file in the *Fusion Compiler script editor*, as in previous labs.
- ✓ Select the menu **Window** ➔ **Clock Tree Analysis Window**.
- ✓ Expand the **SDRAM_CLK** entry in func mode(click on the "+" in front of it) and you will see the two **SD_DDR_CLK*** clocks. Notice that the is_generated column is set to true for these clocks, and their sources are **sd_CK***. Also, you will see the '**M**' and '**G**' symbols in front of the clocks, which identify 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 is applied to its source port sdram_clk. The generated clocks are applied to their source ports sd_CK and sd_CKn, respectively.
- ✓ Perform closer analysis of SDRAM_CLK in func mode by right-clicking on it, then selecting "**Clock Tree Object List**". In the "**Find CTS Objec**t" dialog that appears, select **OK**.



- ✓ In this new window, you will see all valid *sink pins* of this clock. You will not see the output ports sd_CK and sd_CKn in this list.
- ✓ 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.

- ✓ 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 Ctr-F (or click on **Search**...) then enter the search string "**sd_ C**K" 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.

Task 2: Clock Tree Balancing

- ✓ 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, Fusion Compiler needs to be explicitly told about them before CTS.

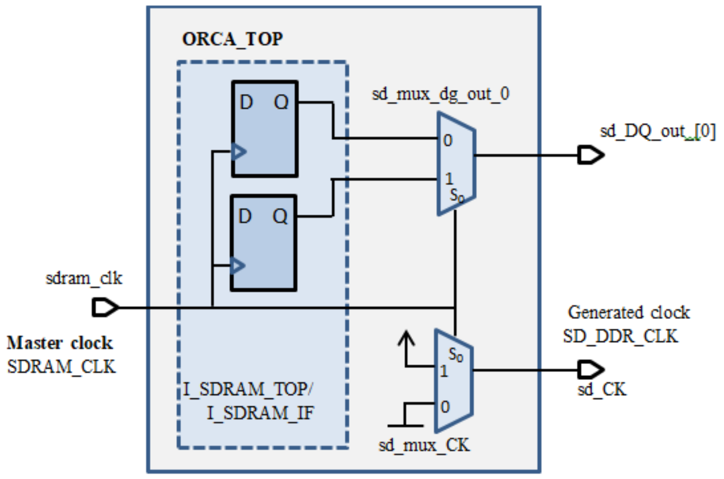


Figure 1. SDRAM interface

- ✓ 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_TOPblock.
- ✓ 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*.
- ✓ In the view window that should still be open, 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.
- ✓ Click on **Dismiss Search** and then **Exit** the view window.
- ✓ Select and run the commands from the **run. tcl** script using the built-in script editor.
- ✓ Apply balancing constraints for the S0 pins in all modes:

```
set_clock balance points \
-modes [a11_modes] \
-balance_points [get_pins "I_SDRAM_TOP/I_SDRAM_IF/sd_mux_ */s0"]
```

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

```
Vreport_clock_ balance_points
```

- ✓ Note that you will find many balance point constraints. These were created bycompile_fusion CCD, which is enabled by default.

- ✓ The balance points you set will be listed below the following lines:
  - Clock Independent :
  - Balance points :
- ✓ The reason the balance points are "Clock Independent" is that you did not specify a clock to go with the exception. If the exception is intended to be balanced concerning a specific clock, and multiple clocks are reaching this point, then a clock should be specified. This is not the case here.
- ✓ Have another look at a *clock structure* report, and search for sd_mux:

```
Vreport_clock_ qor -type structure
```

- ✓ In the view window, search for sd_CK.

  - **Question 2:** How are the **s0** (select) pins of the MUXes labeled now?

    ………………………………………………………………………………………
    ………………………………………………………………………………………
    ………………………………………………………………………………………

  - **Question 3:** Howis the **sd_CK** port labeled now? What does this mean?

    ………………………………………………………………………………………
    ………………………………………………………………………………………
    ………………………………………………………………………………………

- ✓ Close the *view* window.
- ✓ 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_*"]
```

- ✓ Verify your settings with the **report_dont_touch** command:

```
report_don't_touch I_SDRAM_TOP/I_SDRAM_IF/sd mux _*
```

- ✓ 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?

    ………………………………………………………………………………………
    ………………………………………………………………………………………
    ………………………………………………………………………………………

- ✓ 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"]
```

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

```
report_clock_tree_options
```

- ✓ When performing CTS, it is generally desirable to use a specific cell for synthesis, instead of letting *the Fusion Compiler* choose any cell from the library.
- ✓ For example, Cells that help to reduce skew (identical rise/fall ramp times); Cells that 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.
- ✓ 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.
- ✓ 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*_ LVI */INVX*_RVT * \
/CGL* */LSUP* */*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**.
- ✓ Instead of sourcing an edited copy of the **cts_leq_set.tcl** file that you created earlier, you can source the available version:

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

- ✓ Generate a report to ensure that the **correct lib-cell** purpose was indeed set, and don't_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 lab-provided alias **_full_lib_report**.

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**.
- ✓ Open the file **scripts/ndr.tcl** in an editor.
- ✓ 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?
    ……………………………………………………………………………………
    ……………………………………………………………………………………
    ……………………………………………………………………………………

- ✓ Apply the clock NDRs: `source -echo scripts/ndr.tcl`
- ✓ 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.
- ✓ 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 a11*), and the **min/max layer constraints for each clock segment**.

Task 4: Timing and DRC Constraints

- ✓ 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 the **set_driving_cell** of **set_input_transition**?
    ……………………………………………………………………………………
    ……………………………………………………………………………………
    ……………………………………………………………………………………

- ✓ Fix the problem found above by adding a driving cell to the **ate_c1k** 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 chock ports again:

```
set_driving_cell -scenarios [all scenarios] \
-lib_cell NBUFFX16_RVT [get_ports ate_clk]
report_ports -verbose [get_ports *clk]
```

- ✓ 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]
}
```

- ✓ Apply a **max transition** constraint of **0.15ns** on **all clocks**, in **all corners** of the **func mode.**
- ✓ Enable the removal of **clock reconvergence pessimism** to eliminate the timing pessimism of OCV **timing derating** on shared launch/capture clock tree branches.
- ✓ 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 second **##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

- ✓ Build and route the clock trees. Remember to disable CCD, ifs not needed for now:

```
set_app_options -name clock_opt.flow.enable_ccd \
-value false
clock_opt -to route_clock
```

- ✓ 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 to see the NDRs **routingrule** that has been applied.
- ✓ 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.
- ✓ Have a look at the **clock tree latency gra**ph for SDRAM_CLK:
  - In the GUI: **Window** ➔ **Clock Tree Analysis Window**
  - Check the little box in the top right corner next to 'Filter clock by 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).
  - Right-click on SDRAM_CLK under the **func** scenario, then select **ClockTree Latency Graph of selected Corner**.

Lab 08(b) - Running CTS

Objective : *To get familiarized with the clock tree synthesis and Post-CTS optimization capabilities in Fusion Compiler.*

Working Directory : /home/<user_name>/PD_LABS/FC/lab08b/

Project Name : ORCA_TOP.dlib

Learning outcomes:

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

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.
- ✓ Change to the working directory for the lab08b, then load the synthesized and placed design:

```
UNIX%cd lab_8b
UNIX%   fc shell -gui -f load.tcl
```

- ✓ The script will make a copy of the compiled block and open it.
- ✓ Open the **run.tcl** file in the Fusion Compiler script editor, as in previouslabs.
- ✓ Generate a timing QoR summary.

```
report_qor -summary
```

- **Question 1:** From a timing standpoint —is the design ready for CTS? What about the fold violations?

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

- ✓ Set the **current mode** to **“func”**.
- ✓ 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:** Whatis the **source** of the **SD_DDR_CLK** clock?

  …………………………………………………………………………………
  …………………………………………………………………………………
  …………………………………………………………………………………

✓ Generate a **clock skew** report:report_clocks –skew

- **Question 6:** What is the smallest/largest **Setup Uncertainty**?
  What is the smallest/largest **Hold Uncertainty**?

  …………………………………………………………………………………
  …………………………………………………………………………………
  …………………………………………………………………………………

✓ Generate a **clock groups** report:

```
report_clocks -groups
```

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

  …………………………………………………………………………………
  …………………………………………………………………………………
  …………………………………………………………………………………

✓ Generate a clock tree summary report for both modes (default report):

```
v report_clock_gor -modes func
v report_clock_ gor -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?

  …………………………………………………………………………………
  …………………………………………………………………………………
  …………………………………………………………………………………

✓ 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.tcl1). Also, double-check that all scenarios are active.
- ✓ 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
```

- ✓ 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 \
-valué true
```

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

```
set_app options \
-name time.remove_clock_reconvergence_pessimism \
-value true
```

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_optionsa \
-reference_clock PCT_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” anda “Details” section.
- ✓ The summary section will show you how many problems were found of eachproblem category, and

if there is a suggested solution, for example:

- CTS-019 2 None Clocks propagate to output ports
- CTS-905 5 None There are clocks with no sinks

✓ For more details, review the **Details** section: The detailed section for CTS-905 (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 $D_DDR_CLKn (repeated for each mode,func, and test).

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

Task 5: Perform Classic CTS

- **Note:** If you are performing this step after you have already performed CCD, then you will need to re-load the design and re-apply all settings. To simplify this, just restart Fusion Compiler and use the script **scripts/load_a11.tc**l - this script will re-load the design and set up everything up to this point.

✓ Perform classic clock tree synthesis and route the clock trees. Disable CCD, but enable local skew CTS:

```
set_app_options  -name clock_opt.flow.enable_ccd \
-value false
set_app_options -name cts.compile.enable_local_skew \
                        -value true
set_app_options -name cts.optimize.enable_local_skew \
                        -value true
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**.

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

```
report_clock_qor
v report_clock_qor -type local_skew -corner ss_125c
report_clock_qor -type area
report_clock_qor -mode func -corner ss_l25c \
-significant_digits 3
```

✓ Record the various clock QoR statistics:

✓ Several 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			

✓ Another very useful report is the **robustness report**:

```
v report_clock_qor -type robustness -mode func \
-corner ff _m40c -robustness_corner ss_l25c
```

✓ **Muti-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 comer over thelatency 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 corners compared to the reference comer.
- ✓ A robustness value of less than one indicates a sink with less than average latency in the measured corners compared to the reference comer.
- ✓ The largest and smallest robustness value sinks have the worst multi-corner robustness and could cause timing problems in the measured comer.
- ✓ 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.
- ✓ Generate a different skew report using the **clock timing** report: Concentration the **funcmode** 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 Latencynumbers, plus or minus the **clock reconvergence pessimism** (CRP).
- ✓ Record **Skew** and **maximum Latency** for the indicated clocks:

	Skew	Latenct (max)
SYS_2x_CLK		
SDRAM_CLK		

- ✓ One thing to note when reviewing the skew numbers is that compile_fusion has **CCD** enabled by default. Any balance points created by **compile_fusion** are implemented by **clock_opt**. That is why some of these skews might appear large.
  - **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** diiferent?

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

- ✓ Continue with **task 7** to perform **post-CTS optimization** (this will run for ~15 minutes)

Task 6: Concurrent Clock & Data Flow (CCD)

- ✓ In this task, you will use the CCD flow to build the clock trees and optimize the logic.
- ✓ CCD will take much longer to run compared to classic CTS.
- ✓ If you are performing this step after you have alreadyperformed Classic CTS, then you will need to re-load the design and re-apply all settings. To simplify this, just restart FusionCompiler 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. In other-case ignore this step and continue from the next step.
- ✓ Source the following script - this will make things a little more interesting forCCD, by introducing a few **setup timing violations**, which will need to be fixed by the CCD algorithms. Have a look at a timing summary afterward:

```
source scripts/margins_for_ccd.tcl
```

```
report_qor -summary
```

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

	WNS	TNS	NVE
Setup			
Hold			

- ✓ Ensure that the CCD flow is enabled.

```
reset_app_options clock_opt.flow.enable_ ccd
```

- ✓ **Runclock_opt**to the **route_clock** stage.

```
clock_opt -to route_clock
```

- ✓ After completing the run, record the design QoR to see whether CCD wasable to fix all the artificially introduced violations.
- ✓ Continue with **task 7** to perform **post-CTS optimization** (this will run for ~15 minutes)

Task 7: Perform Post-CTS Optimization

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

- ✓ Set up the design for signal routing, since **global routing** (GRE) will be performed during the **final_opto** stage of **clock_opt.**

```
source scripts/route_setup.tcl
```

- ✓ Execute post-CTS optimization: (This will take a while)

```
clock_opt -from final _opto
```

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

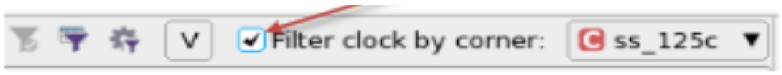
  - **Question 13:** Are there any setup or hold violations left?

…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

- ✓ Confirm that the design has no congestion issues.
- ✓ Save the block and continue to Task 8.

Task 8: Analysis

- ✓ Have a look at the synthesized clock tree using the **clock abstract graph** GUI.In the GUI select **Window→ Clock Tree Analysis Window**.
- ✓ In the new **CTS window**, check the box in the top right banner next to '**Filterclock by corner**'.


Filter clock by corner: ss_125c

- ✓ This allows us to analyze clock latencies, which vary by corner. Make sure the ss_125c corner is selected.
- ✓ 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 **ClockTree Latency Graph** of selected Corner.
- ✓ This screenshot shows the latency graph for SYS_2x_CLK for the classicCTS flow (the CCD

screenshot is shown on the next page):

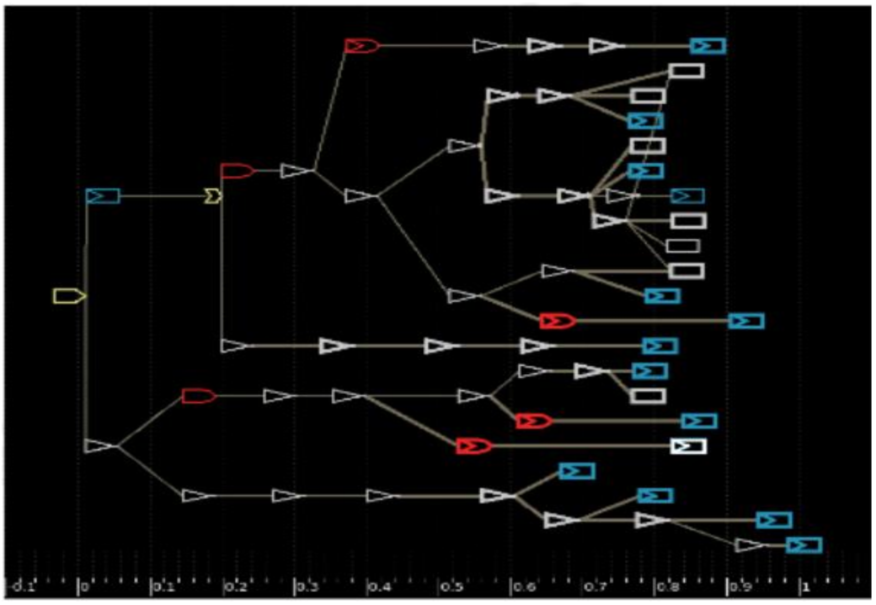


Fig: Latency Graph for SYS_2x_CLK for the classicCTS flow

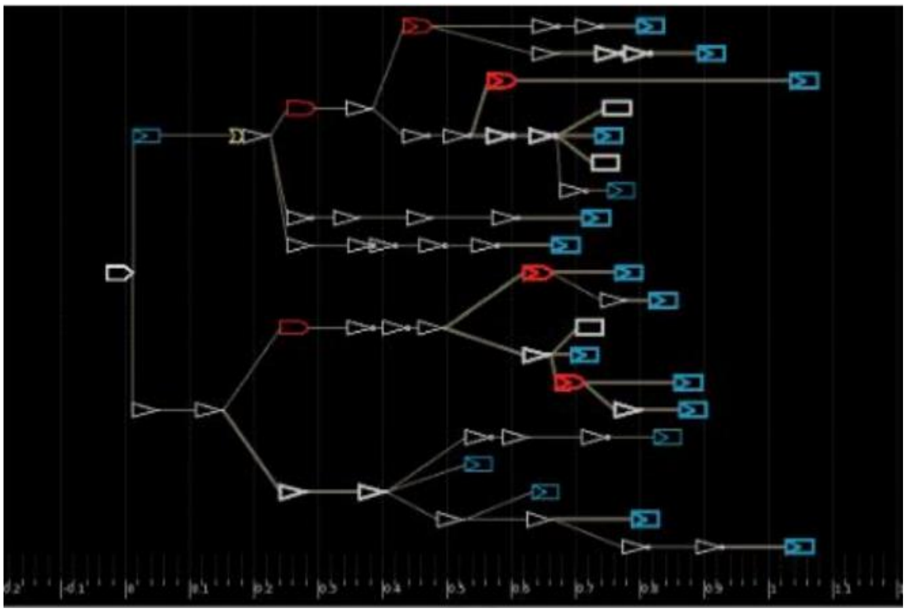
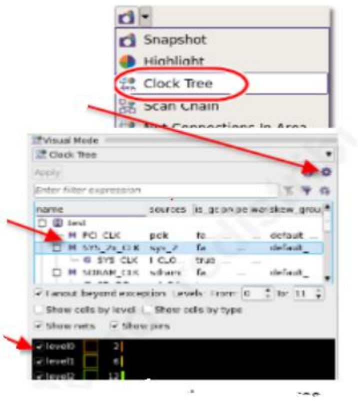


Fig: Latency Graph for SYS_2x_CLK for the classic CCD flow



- ✓ As expected, the latency distribution to the register endpoints will be larger forCCD than for classic CTS
- ✓ Close the CTSWindow
- ✓ Examine the clock tree routing topology, by selecting the **Clock Tree visual mode**.
- ✓ In the panel, click on the little gear symbol 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 7 is the net that is driven by the first driver, etc.

- ✓ Close the **Clock Tree visual mode** panel.
- ✓ 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**?

…………………………………………………………………………………………
…………………………………………………………………………………………
…………………………………………………………………………………………

Lab 09 - Routing and Post-Route Optimization

Objective : *To get familiarized with the Routing and post-route optimization in Fusion Compiler*

Working Directory : /home/<user_name>/PD_LABS/FC/lab09/

Project Name : ORCA_TOP.dlib

Learning outcomes:

- ✓ Perform routeability checks on a placed designwith 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.

Task 1: Load and Check the Post-CTSdesign

- ✓ Invoke Fusion Compiler from the lab_9 directory and load the post-CTS design

```
UNIX% cd lab_9
UNIX% fc_shell -gui -f load.tcl
```

  The script will make a copy of the clock_opt block and open it.
- ✓ To generate the timing QoR summary use the command:

```
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 FC 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. (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**
- ✓ **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 ICC II)

```
define_antenna_rule
define_antenna_layer_rule
define_antenna_aréa_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**.
- ✓ To check the Antenna effect use the commands:

```
source -echo ../ref/tech/saed32nm_ant_1p9m.tcl
report_app_options route.detail.*antenna*
```

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 assigns stage only.
- ✓ In order for **crosstalk prevention** to occur, crosstalk (or signal integrity) analysis must also be enabled.
- ✓ 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.

Routing, DRC Analysis

- ✓ At this stage, we will route all signal nets that have not been routed previously.
- ✓ Any signals that have been routed already (clocks, secondary PG) will not be touchedagain if they are DRC clean.
- ✓ Auto-routing runs **track assignment** and **detail routing.**
- ✓ **Global routing** was already completed during the **clock_opt final_opto** stage.

  - **Question 3:** How many detail route iterations are run by default? (*Hint: Review the man page for route_auto*)

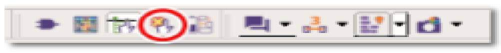
    ………………………………………………………………………………
    ………………………………………………………………………………
    ………………………………………………………………………………

  - **Question 4:** How do you change the default, and how do yu force the router to run through all iterations even though the router might not see any improvements? (*Hint: report_app_options *itreat**)

    ………………………………………………………………………………
    ………………………………………………………………………………
    ………………………………………………………………………………

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 **Sl-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

- ✓ For best correlation with PrimeTime, you should enable **PT delay calculation**, as well as **StarRC fusion extraction**.
- ✓ For timing analysis, you can also use path-based analysis, which is much more accurate at this stage of the design.
- ✓ 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 to enable CCD and Power optimization.

Lab 10 - Signoff

Objective : *Explore the signoff tools and run ECO.*

Working Directory : /home/<user_name>/PD_LABS/FC/lab10/

Project Name : ORCA_TOP.dlib

Learning outcomes:

- ✓ Run ECO Fusion to fix the remaining violations usingPrimeTime+StarRC
- ✓ Perform sign-off DRC checking and fixing
- ✓ Insert standard cell fillers
- ✓ Add sign-off metal fill using ICV

Task 1: Load and Check the Post-Route Design

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

```
% cd lab_10
%  fc_shell -gui  -f  load.tcl
```

- ✓ The script will make a copy of the route_opt block and open it.
- ✓ Generate a timing QoR summary: `report_qor  -sumary`
- ✓ Note down WNS and TNS numbers.

Task 2: Perform Signoff Operations

- ✓ 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 scripteditor) 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 optionthat 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.

ECO Fusion

- ✓ After completing **route _opt**, its important to analyze **signoff** timing in PrimeTimeusing Fusion Extraction parasitic extraction. Any violations uncovered by PT can befixed using PT's physical ECO.
- ✓ Using Fusion, the entire ECO process can be controlled from within FusionCompiler.
- ✓ Review the run.tcl file for the necessary commands and perform one round ofECO fixing, For ECO Fusion, you will require Fusion Compiler, StarRC, and primetime SI.
  - **Note:** After the ECOs have been implemented, you have to analyze timing usingthe command **check_pt_qor**. You should not run Fusion Compiler'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.
- ✓ lf 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 tech file and the ICV DRC runset**.

- ✓ You can review the errors generated by ICV using the error browser. If the errorbrowser 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 veryend of the **run.tcl** script.
- ✓ Examine the metal fill using the GUI, as described in the lecture