

1

IC Compiler™ Data Setup & Basic Flow

Learning Objectives

This lab has two purposes:

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

After completing this lab, you should be able to:

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



Lab Duration:
60 minutes

Introduction

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

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

Answers / Solutions

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

Relevant Files and Directories

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

lab1_data_setup/

.synopsys_dc.setup

Read by IC Compiler upon startup

design_data/

RISC_CHIP.v

RISC_CHIP verilog gate level netlist.

RISC_CHIP.def

RISC_CHIP floorplan in DEF.

RISC_CHIP.sdc

RISC_CHIP timing constraints.

scripts/

opt_ctrl.tcl

Timing and optimization controls.

zic_timing.tcl

A script used to check zero-interconnect timing constraints.

derive_pg.tcl

Create logical P/G connections.

.solutions/

run.tcl

A run script with all the commands executed in this lab.

Instructions

Task 1. Create a *Milkyway* library

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

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

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

2. Use a UNIX text editor or viewer to look at the contents of the `.synopsys_dc.setup` file.
3. At the bottom of the file, we have created the following user-defined variables to help document and simplify the data setup process:

```
#-----
# RISC_CHIP setup variables
#-----

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

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

Lab 1

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

```
lappend search_path ../ref/db ../ref/tlup
set_app_var target_library "sc_max.db"
set_app_var link_library "*sc_max.db io_max.db \
                           ram16x128_max.db"

set_min_library sc_max.db -min_version sc_min.db
set_min_library io_max.db -min_version io_min.db
set_min_library ram16x128_max.db -min_version \
                           ram16x128_min.db
```

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

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

5. Exit the text editor or viewer.
6. Start IC Compiler from the UNIX prompt:

```
UNIX% icc_shell
```

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


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

```
printvar sdc_file
```

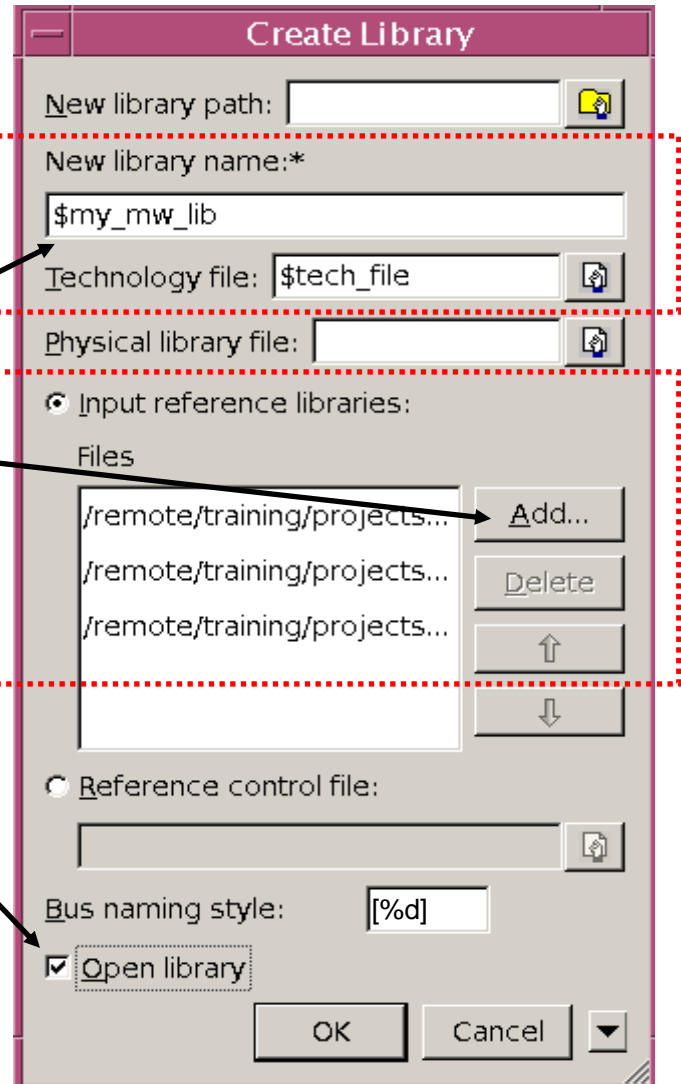
8. Start the GUI. The *MainWindow* will appear after a short while:

```
start_gui
```

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

9. Create the design library:
 - a. Use the *MainWindow* menu **File → Create Library ...** to bring up the *Create Library* dialog box.
 - b. Use the *variables* already defined for the library and tech file names.
 - c. Attach *reference libraries* to your design library:
Click the **Add...** button.
Double-click  to move up one level, then double-click *ref* followed by *mw_lib*.
 - d. Add the “*io*” and “*ram16x128*” libraries as well.
 - e. Select the “**Open library**” check box to open the design library after it is created.
 - f. Click **OK**.

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



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

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

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

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

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

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

1. Before reading in the Verilog netlist make sure the design library is open: An easy way to do so is by checking if the **File → Open Library ...** entry is grayed out. If it is, this confirms that a design library is currently open.
2. Select **File → Import Designs ...** to bring up the *Import Design* dialog box.
3. Under *Input format* select *verilog*.
4. Click **Add ...** then browse to select the file **design_data/RISC_CHIP.v** and **Open**.
Under *Top design name* enter **\$top_design (or RISC_CHIP)**.
Click **OK**.

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

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



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

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

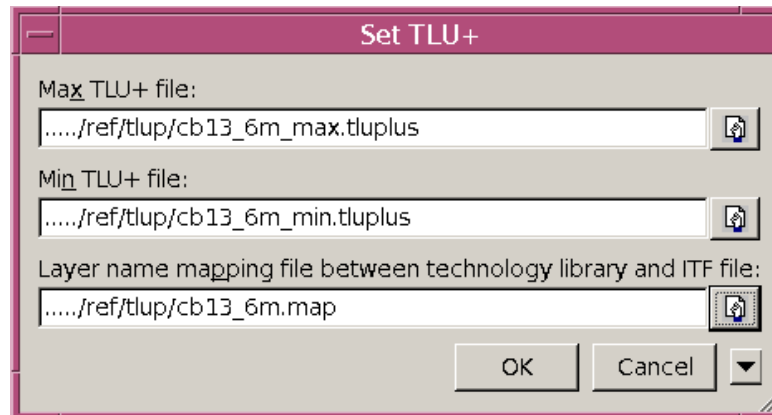
.....
.....

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

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

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

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



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

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

- Check the physical and logical libraries for consistency:

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

```
check_library
```

There are two messages that are of interest:

“Number of cells missing in logic library : 19”

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

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

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

Lab 1

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

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

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

```
check_tlu_plus_files
```

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

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

```
list_libs
```

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

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

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

There should be no “unconnected” power or ground pins.

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

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


10. Apply the top level design constraints:

```
read_sdc $sdc_file
```

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

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

```
check_timing
```

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

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

```
report_timing_requirements
```

Note: There should be no timing exception constraints reported.

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

```
report_disable_timing
```

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

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

```
report_case_analysis
```

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

Lab 1

15. Verify that the clocks are appropriately modeled:

```
report_clock
report_clock -skew
```

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

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

.....

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

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

```
source $ctrl_file
```

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

```
source sc[TAB]z[TAB]

# The above file scripts/zic_timing.tcl contains:
#   set_zero_interconnect_delay_mode true
#   redirect -tee zic.timing { report_timing }
#   set_zero_interconnect_delay_mode false
```

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

```
exec cat zic.timing
```

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

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

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

.....

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

```
remove_ideal_network [get_ports scan_en]
```

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

```
save_mw_cel -as RISC_CHIP_data_setup
```

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

Task 3. Basic Flow: Design Planning

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

1. Read in the provided *DEF* file:

```
read_def $def_file
```

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

2. Press [F] in the *LayoutWindow* to refresh the view. You should now see the floorplanned design.
3. Ensure that standard cells will not be placed under the power and ground metal routes (this constraint is not part of *DEF*):

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

Lab 1

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

```
save_mw_cel -as RISC_CHIP_floorplanned
```

Task 4. Basic Flow: Placement

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

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

Question 4. Does the placed design meet timing?


.....

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

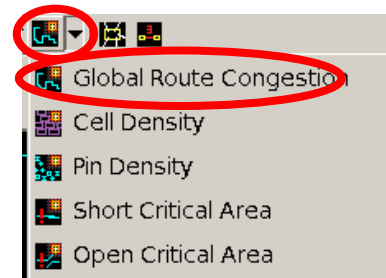
Question 5. Would you call this a “core limited” design?

.....

Analyze congestion :

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

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



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

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

Click “**OK**”.

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

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

4. Save the design cell:

```
save_mw_cel -as RISC_CHIP_placed
```

Task 5. Basic Flow: CTS

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

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

Question 6. Is the timing still OK?

.....

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

Click “**Reload**”.

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

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

Click **OK**.

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

3. Remove the clock tree highlight by closing (“**x**”) the visual mode window.
4. Save the design cell:

```
save_mw_cel -as RISC_CHIP_cts
```




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

Task 6. Basic Flow: Routing

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

1. Invoke IC Compiler's GUI:

```
UNIX% icc_shell -gui
```

2. Since the design library has already been created, and you saved the layout cell after CTS, all you have to do is load the *RISC_CHIP_cts* cell from the *risc_chip.mw* design library, as follows:
 - a. In the *MainWindow* click on the little yellow “open design” icon  on the top left, or use the menu command: **File → Open Design ...**
 - b. In the *Open Design* dialog panel, click the yellow folder icon . The *Select Library* dialog box opens. Select the  library folder *risc_chip.mw* and click **Choose**.
 - c. Select *RISC_CHIP_cts* and click **OK** to open it.
3. Re-apply the timing and optimization controls, which were applied during data setup. This is required because some of the settings are applied using variables. In general, variable settings are not saved with the design cell – they remain set during the current IC Compiler session. After exiting and re-invoking IC Compiler, the variables are reset to their original default values:

```
source $ctrl_file
```

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

```
route_opt
```

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

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

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

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

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

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

```
v rt -delay min
```

7. Generate physical design statistics:

```
report_design -physical
```

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

8. Save the design.

```
save_mw_cel -as RISC_CHIP_routed
```

9. Quit the IC Compiler shell

```
exit      or      quit
```

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



Answers / Solutions

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

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

Question 2. What is the modeled combined effect of skew, jitter and margin for setup time?

(Hint: “Minus Uncertainty” relates to *setup* time)

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

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

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

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

Question 4. Does the placed design meet timing?

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

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

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

Question 6. Is the timing still OK?

Yes, all paths have positive slack.