IC Compiler

Lab 2: IC Compiler Basic Flow

Objectives

This lab has two purposes:

- Walk you through the process of creating and maintaining a Milkyway database to hold your design data.
- Run a complete basic P&R flow.

After completing this lab, you should be able to:

- Create a Milkyway database for your design
- Attach reference libraries to your design library
- Attach TLU+ models to your library for enhanced timing analysis accuracy
- Read the verilog netlist from synthesis
- Create a basic floorplan
- Place and optimize the design using the place_opt command
- Create a clock tree for the design using the clock_opt command
- Route and optimize the design using the route_zrt_auto command
- Perform chip finishing step

As was discussed in classes, we can run ICC in batch mode, shell interactive mode and GUI mode. Batch mode means that we will write in a script file all the commands needed for the complete flow, and then we will call that script from ICC. In shell interactive mode, we execute all the required commands one by one through ICC command line. Finally, GUI mode allows the user to execute commands using a graphical interface. Batch mode is useful when the designer has experience with the tool, and wants a quick way to execute the whole flow and just evaluate the final output. This time we will run ICC in shell interactive mode, to see how the chip evolves in each step.

Activity 1: Create a setup file Milkyway database for top

Although we want to run ICC in interactive mode, it is recommended to have a setup file for the setup step. This file will contain all the commands that define ICC environment, i.e. search path, logical libraries, MW reference libraries, etc. Instead of executing all these commands each time we open an ICC session, is better to just source one script that has everything.

1. Go to project directory and create an empty file called scripts/setup.tcl using your favorite text editor. Use the cheat-sheet provided in the class to know which commands and variables should be defined. The input data files you will need are listed in the following table.

Input data	**Name**
Target library	c5n_utah_std_v5_t27.db
Link library	c5n_utah_std_v5_t27.db io.db
Search path	MW/ logic/ scripts/ inputs/
MW reference libraries	MW/UTAH_V1P1 MW/ICG MW/IO
Technology file	MW/UTAH.tf
TLU+ file (max & min)	MW/ami500.tluplus
Mapping file	MW/ami500hxkx_3m.map

2. Invoke ICC by typing `icc_shell` in the unix shell. Once open, load library setup tcl script.

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

Activity 2: Import the design and apply a floorplan

1. In the next step you will import your synthesized verilog netlist and link all its components together. First make sure the design library is open (`MW_TOP_LIB.mw`). One easy way to do this is to go to **File → Open Library**. If the entry is grayed out, the library is open.

2. Import the synthesized netlist by selecting **File → Import Designs** or by typing the command below.

```
import_designs inputs/mapped.ddc -format ddc
```

Note: You should notice that a new *LayoutWindow* was opened, displaying the physical *CEL* view of your design named top. Since the design has no floorplan yet, and nothing has been placed, what you are seeing is all the standard cells (pink rectangles) from your netlist stacked on top of each other at the origin (0,0).

3. Verify that the design is constrained for timing:

```
check_timing
```

Note: Note that no SDC file has been read. (Why?) If there we missing constraints, the command would return WARNING messages letting you know which clock(s) or output(s) were unconstrained. You should not see any warnings about missing constraints. **If you do see, it's a good idea to come back to synthesis step and check.**

4. Before placing the standard cells, a *design plan* must be defined for the design, so that the placer knows where the standard cells can be placed. The design plan includes the *floorplan* of the core placement area and periphery, as well as the *power plan*. Execute the following script to build your design's floorplan and power plan.

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

In a separate UNIX shell, open this script using a text editor and study the commands (check the man pages), and put comments after each one to explain its functionality. In general, floorplanning is a complex process, so take this just as a basic example.

Fit the design to the window **[F]**.

Note: You should now see the design plan in the lower-left area of the *LayoutWindow*. The as yet unplaced standard cells are stacked in a tall column to the right of the design plan. Zoom in and take a look at the design plan. It consists of a square placement *core area*, surrounded by the *I/O pads*. In the core area there are many horizontal *placement rows*, each containing a horizontal VDD and VSS *power rails.* Furthermore, there are two sets of vertical VDD/VSS *straps* on M3 (red). This ensures a well-distributed power network for the block.

5. Verify that the libraries and the design plan are consistent with the design requirements.

```
check_physical_constraints
```

6. It is highly recommended to save your design at various key stages along the way. We have just completed the design planning stage, so let's save the design.

```
save_mw_cel -as top_postFloorplan
```

Note: The *CEL* view in the *LayoutWindow* has not changed – it is still the original *top.CEL*. The above command has created a new CEL view, but it does not open that view. To do so you would need to first close the current cell with `close_mw_cel`, and then open the new one with `open_mw_cel`. We will continue working on the original *top.CEL*.

Activity 3: Perform placement, CTS and routing

1. Place the standard cells and verify that the design meets timing.

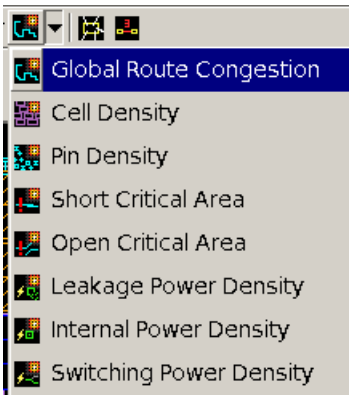
```
place_opt
report_timing > reports/top.timing.placement.rpt
```

Note: The > operator redirects output of the command (e.g. `report_timing`) to the specified file.

Note: `place_opt` has various options. Check them on the class's slides or execute man `place_opt`.

2. Examine the placement and floorplan in the Layout Window. You should now see that the standard cells have been placed inside the core area.

3. Bring up a congestion map. Select "Global Route Congestion" from the dropdown, select "Reload" and "OK". The congestion map will be shown in the Layout. There should be little to no congestion. *It is always a good idea to examine congestion…*



4. Close the congestion map by clicking on the small "x" in the upper right corner of the Global Route Congestion panel.

5. Save the design.

```
save_mw_cel -as top_postPlaceOPT
```

6. Next you will perform clock tree synthesis. Since we want CTS to calculate the actual clock skews rather than the estimated value, we need to remove the clock uncertainty from all clock objects. Generate a post-CTS timing report.

```
remove_clock_uncertainty [all_clocks]
clock_opt –fix_hold_all_clocks

# Route PG rails for new cells (CTS/PSYNOPT)
derive_pg_connection -power_net {vdd!} -ground_net {gnd!} -
verbose
preroute_standard_cells -nets {vdd! gnd!} -connect horizontal
report_timing > reports/top.timing.cts.rpt
report_clock_tree
```

Note: Timing should still be OK.

7. Take a look at the layout window. You should be able to see the metal routes associated with the clock tree. You may need to zoom in a little.

8. Generate another congestion map to ensure that there is no significant congestion.

9. Save the design.

```
save_mw_cel -as top_postCTS
```

10. Route the design. This will take route all the remaining signal nets.

```
route_opt
```

Take a look at the layout window once again. You should be able to see the metal routes associated with different signals. Pay attention to the fact that standard cells are very scattered, having gaps between contiguous cells.

11. Save the design.

```
save_mw_cel -as top_postRouteOpt
```

Activity 4: Perform chip finishing

1. Insert Filler cells. All gaps between standard cells will be filled

```
insert_stdcell_filler -cell_with_metal "FILL8 FILL4 FILL2
FILL"  -connect_to_power "vdd!" -connect_to_ground "gnd!"
derive_pg_connection -power_net {vdd!} -ground_net {gnd!} -
verbose
preroute_standard_cells -nets {vdd! gnd!} -connect horizontal
```

The first command inserts the actual cells. `derive_pg_connection`, and `preroute_standard_cells` are executed to fix possible problems created by filler insertion (short circuits, open circuits, etc.)

2. Fix antenna violations:

```
set_route_zrt_detail_options -default_gate_size 1.8 -
default_port_external_gate_size 1.8
define_antenna_rule MW_top_LIB -mode 1 -diode_mode \
  0 -metal_ratio 1000 -cut_ratio 0
route_zrt_detail -incremental true
verify_zrt_route -antenna true
insert_metal_filler  -fill_poly -bounding_box \
  { { 260 260 } { 3460 2626 } }  -out self  \
  -timing_driven  -to_metal 3
# Layer fill
set_parameter -name wellFillerAlignWithCell -value 1 -module
apl
insert_well_filler -layer NWELL -ignore_PRboundary
save_mw_cel -as TOP_postChipFinish
```

3. Take a look at your final, complete, design layout.

4. Take a look to scripts/icc.tcl. This script has almost everything we covered in this lab. Go to the signoff part and execute the commands from there. As a starting point, run DRC and LVS checkings.

```
verify_lvs
verify_pg_net
```

Take a look to the reports and check if there are errors. You can also do these checks in the GUI. Go to "*Verification*" tab in the main menu and then select "*LVS.*." or "*DRC.*."

5. Save the design.

```
save_mw_cel -as top_final
```

6. Close the library and quit the IC Compiler shell.

```
close_mw_lib
exit
```

In summary, starting from mapped netlist and constraints (outputs of Design Compiler), you used IC Compiler to create a design plan, place the design, perform clock tree synthesis, and route the entire design, ending up with a complete layout of your design. This concludes the workshop.

Cheat-sheet for setup file

Command	Description
`set_app_var search_path <your_search_path>"`	Defines the search path (where ICC will look for files). You must replace *<your_search_path>* with the actual directories.
`set_app_var target_library "<your_libs>"`	Define the logical libraries (logical models) of the standard cells for insertion purposes. Put here your .db files.
`set_app_var link_library "* <your_libs>"`	Define the logical libraries (logical models) of the standard cells and macro cells for linking purposes. Put here you r .db files.

`create_mw_lib -technology <tech_file> \`
`-mw_reference_library <your_mw_libs> \`
`<your_design_lib_name>`
	Creates de Design Library (the container). You must specify the location of the Tech File and the Milkyway References LIbraries

`open_mw_lib <design_lib>`	Opens a design library

`set_tlu_plus_files -max_tluplus <tech_file> \`
`-min_tluplus <tech_file> \`
`-tech2itf_map <map_file>`
	Sets the TLU+ files to use for net delay calculation

`check_tlu_plus_files`	Checks correctness of TLU+ files
`check_library`	Checks correctness of logic and MW libraries

ICC Script for Basic Flow (scripts/icc.tcl)

```
################################################################################
# Script: Place & Route script for OnSemi C5N Process
################################################################################
source -echo scripts/setup.tcl

# Verify librery consistency
set_check_library_options -logic_vs_phy
check_library

################################################################################
# Source Input Data Base
################################################################################
# Load Mapped DDC
import_design inputs/top_mapped.ddc -format ddc

################################################################################
# Design Planning
################################################################################
source scripts/floorplan.tcl
write_def -rows_tracks_gcells -pins -macro -output results/fp.def
save_mw_cel -as toppostFloorplan

################################################################################
# Placement
################################################################################
place_opt
save_mw_cel -as top_postPlaceOPT

################################################################################
# Clock Tree Synthesis
################################################################################
remove_clock_uncertainty [all_clocks]
clock_opt -fix_hold_all_clocks

# Route PG rails for std cells (CTS/PSYNOPT)
derive_pg_connection -power_net {vdd!} -ground_net {gnd!} -verbose
preroute_standard_cells -nets {vdd! gnd!} -connect horizontal

################################################################################
# Routing
################################################################################
route_opt

# Account for new buffers inserted
derive_pg_connection -power_net {vdd!} -ground_net {gnd!} -verbose
preroute_standard_cells -nets {vdd! gnd!} -connect horizontal
route_zrt_detail -incremental true

save_mw_cel -as top_postRouteOpt
```

```
################################################################################
# Chip Finish
################################################################################
# Std Cell Fillers (Provide NWELL & rail continuity)
insert_stdcell_filler -cell_with_metal "FILL8 FILL4 FILL2 FILL" \
  -connect_to_power "vdd!" -connect_to_ground "gnd!"
derive_pg_connection -power_net {vdd!} -ground_net {gnd!} -verbose
preroute_standard_cells -nets {vdd! gnd!} -connect horizontal

# Antenna Constraints (Fix antenna violations for gate driving nets)
set_route_zrt_detail_options -default_gate_size 1.8 -default_port_external_gate_size 1.8
define_antenna_rule $mw_design_library -mode 1 -diode_mode 0 -metal_ratio 1000 -cut_ratio 0
route_zrt_detail -incremental true
verify_zrt_route -antenna true

# Fill to meet density rule
insert_metal_filler  -fill_poly -bounding_box { { 260 260 } { 3460 2626} }  \
  -out self  -timing_driven  -to_metal 3

# Layer fill
# This one makes the well to align preventing DRC
set_parameter -name wellFillerAlignWithCell -value 1 -module apl
insert_well_filler -layer {NWELL} -ignore_PRboundary -higher_edge min -lower_edge max

save_mw_cel -as top_postChipFinish

################################################################################
# Signoff
################################################################################
verify_pg_nets
verify_lvs

################################################################################
# Output Data
################################################################################
# Dump out Verilog netlist for post P&R gate level Sim and/or STA
change_names -rules verilog -hierarchy
write_verilog results/top.pnr.v
write_verilog -no_physical_only_cells  results/top.pnr.no_phys_only.vg
write_sdc     results/top.pnr.sdc

# Store parasitics annotation file
extract_rc -coupling_cap
write_parasitics -format SBPF -output "results/top.pnr.sbpf"
write_sdf results/top.pnr.sdf

report_clock_tree > results/top.clock
report_timing > results/top.timing

# Dump out GDS (26: pad 52: glass cut)
set_write_stream_options   -output_filling {fill} -output_pin {text geometry} -keep_data_type -
child_depth 20  -map_layer MW_layer_map
write_stream -lib_name MW_top_LIB -format gds results/top.final.gds
```