3 Multi-Corner Multi-Mode and Strategies

Learning Objectives

During this lab, you will study Design Library setup, Multi Corner Multi Mode setup, logic_opto stage of compile_fusion and effort level control for timing/power/area optimization.

After completing this lab, you should be able to:

- Design Library setup
- Create Multi Corner Multi Mode constraints
- Implement logic_opto stage
- Controll efforts

Lab Duration: 40 minutes

Lab Instructions

Lab work structure

Lab directory organized as follows:

```
FC_Labs/labs/lab3_mcmm_and_strategies/
    scripts/
          logic_opto_mcmm_basic.tcl
          logic_opto_mcmm_effort_for_area.tcl
          logic_opto_mcmm_effort_for_power.tcl
          logic_opto_mcmm_effort_for_timing.tcl
    work/
          logs/
          Makefile
```

Move to work directory:

```
cd FC_Labs/labs/lab3_mcmm_and_strategies /work
```

This lab work consists of 4 parts:

1. Basic synthesis till logic_opto stage with MCMM
2. Synthesis till logic_opto stage with MCMM and high area effort
3. Synthesis till logic_opto stage with MCMM and power optimization
4. Synthesis till logic_opto stage with MCMM and high timing effort

You can run this lab by using Makefile in work directory. To run all 3 steps, use the following command in terminal:

```
make all
```

To run each step separately with Makefile, use the following commands in terminal:

```
make run_1
make run_2
make run_3
```

```
make run_4
```

Or invoke the Fusion Compiler and source the **corresponding** script:

```
fc_shell  -no_log -f ../scripts/logic_opto_mcmm_basic.tcl
| tee ./logs/logic_opto_mcmm_basic.log

fc_shell  -no_log -f
../scripts/logic_opto_mcmm_effort_for_area.tcl | tee
./logs/logic_opto_mcmm_effort_for_area.log

fc_shell  -no_log -f
../scripts/logic_opto_mcmm_effort_for_power.tcl | tee
./logs/logic_opto_mcmm_effort_for_power.log

fc_shell  -no_log -f
../scripts/logic_opto_mcmm_effort_for_timing.tcl | tee
./logs/logic_opto_mcmm_effort_for_timing.log
```

In case you want to rerun the lab, you can clean your working area by using the following command in terminal:

```
make clean
```

At the beginning of lab work you **must** source the following setup scripts in Fusion Compiler to correcly setup the flow:

```
#### Sourcing common setup script

source -echo ../../setup/fc_common_setup.tcl

#### Sourcing flow setup script

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

Design Library Setup

Before starting the full synthesis flow, some data need to be provided in the Design Library.

1. Need to define the default cell site in the design. Site definition comes from the **Tile** section of the Technology File. There may be several sites available in the Technology File, so the user must specify the default one. To set the site name "unit" as default, use the following command:

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

   The site symmetry also must be specified as in the below command:

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

   The Y value means that upper and lower site rows are flipped.

2. Routing directions must be specified for the available metal layers from the Technology File for correct routing track creation and further routing. Use the below commands to set vertical and horizontal directions for the corresponding metal layers.

```
set_attribute [get_layers {M1 M3 M5 M7 M9}] \
routing_direction vertical
set_attribute [get_layers {M2 M4 M6 M8 MRDL}] \
routing_direction horizontal
```

3. Parasitic information must be provided for correct tool work and analysis. The parasitic information can be set by using **tlup** or **nxtgrd** files. You can specify several tlup or nxtgrd files for various corners. The example commands for reading the parasitic information with tlup format into the design library are shown below:

```
read_parasitic_tech -layermap ${LAYER_MAP_FILE} \
    -tlup ${TLUP_MAX_FILE} -name maxTLU
read_parasitic_tech -layermap ${LAYER_MAP_FILE} \
    -tlup ${TLUP_NOM_FILE} -name nomTLU
read_parasitic_tech -layermap ${LAYER_MAP_FILE} \
    -tlup ${TLUP_MIN_FILE} -name minTLU
```

   To read the nxtgrd files, just provide nxtgrd file instead of tlup file in the above commands.

The above setup is done in **FC_Labs/labs/setup/tech_setup.tcl** script.

Multi Corner Multi Mode Setup

Fusion Compiler supports Multi Corner Multi Mode (MCMM) setup and design optimization. The optimization process is performed concurrently for different metrics for active scenarios. MCMM setup allows you to create different corners, modes and scenarios to perform analyzes and optimization. For each scenario, you can provide its own unique constraints besides the global ones.

1. Use the below commands to remove any existing corners, modes and scenarios definitions:

```
remove_corners   -all
remove_modes     -all
remove_scenarios -all
```

2. To create corners in Fusion Compiler use the following commands:

```
create_corner Fast
create_corner Typical
create_corner Slow
```

The above commands create 3 corners named Fast, Typical and Slow for further analysis and optimization.

3. Then the corresponding parasitics information must be set for appropriate corners. For that use the below commands:

```
set_parasitics_parameters -early_spec minTLU \
    -late_spec  minTLU -corners {Fast}
set_parasitics_parameters -early_spec nomTLU \
    -late_spec  nomTLU -corners {Typical}
set_parasitics_parameters -early_spec maxTLU \
    -late_spec  maxTLU -corners {Slow}
```

4. To create mode named FUNC use the following command:

```
create_mode FUNC
```

5. After corner and mode creation, the corresponding scenarios can be created. Below commands create 3 scenarios named FUNC_Fast, FUNC_Typical and FUNC_Slow:

```
create_scenario -mode FUNC -corner Fast    -name FUNC_Fast
create_scenario -mode FUNC -corner Typical \
    -name FUNC_Typical
create_scenario -mode FUNC -corner Slow    -name FUNC_Slow
```

6. You can control the current corner, mode and scenario by using the below commands:

```
current_corner ${CORNER_NAME}
current_mode ${MODE_NAME}
current_scenario ${SCENARIO_NAME}
```

7. To read constraints for each scenario, use the following script:

```
foreach scenario ${scenarios} {
    current_scenario ${scenario}
    read_sdc ${SDC_FILE}
}
```

8. Constraints are used for design optimization. Below constraints are used for this laboratory work:

```
set PERIOD  2
set CLOCK_NAME wb_clk_i
set TRANSITION 0.2
set UNCERTAINTY 0.3

create_clock -period $PERIOD -name $CLOCK_NAME \
    [get_ports $CLOCK_NAME]
set_clock_uncertainty -setup $UNCERTAINTY \
    [get_clocks $CLOCK_NAME]
set_clock_transition -rise $TRANSITION \
    [get_clocks $CLOCK_NAME]
set_clock_transition -fall $TRANSITION \
    [get_clocks $CLOCK_NAME]
```

A clock signal named **wb_clk_i** is created with 2ns clock period. Clock signal's rise and fall transitions are defined with clock uncertainty.

9. For each scenario, the corresponding operating conditions must be specified as in the below script:

```
current_corner Fast
current_scenario FUNC_Fast
set_operating_conditions ${PVT_FF}

current_corner Typical
current_scenario FUNC_Typical
set_operating_conditions ${PVT_TT}

current_corner Slow
current_scenario FUNC_Slow
set_operating_conditions ${PVT_SS}
```

Operating conditions provide the information about process, voltage and temperature used for the design optimization and analysis.

10. Finally, created scenarios must be configured for different types of analyzes. Configuration can be done as in the below script:

```
set_scenario_status FUNC_Fast -setup false -hold true \
    -leakage_power false -dynamic_power true  \
    -max_transition false -max_capacitance true  \
    -active true
set_scenario_status FUNC_Typical -all -active true
set_scenario_status FUNC_Slow -setup true  \
    -hold false -leakage_power true -dynamic_power true  \
    -max_transition true -max_capacitance false  \
    -active true
```

Above setup is done in **FC_Labs/labs/setup/mcmm_setup.tcl** script.

logic_opto stage

The Fusion Compiler tool performs unified physical synthesis and supports complete RTL_to_GDS flow. Fusion Compiler combines traditional logic synthesis with floorplan creation, placement and optimization.

logic_opto stage includes logic-based delay optimization and automatic floorplan creation**.** Physical synthesis aspects will be covered in next lab works.

```
compile_fusion -to logic_opto
```

Block view after logic_opto stage is presented below:

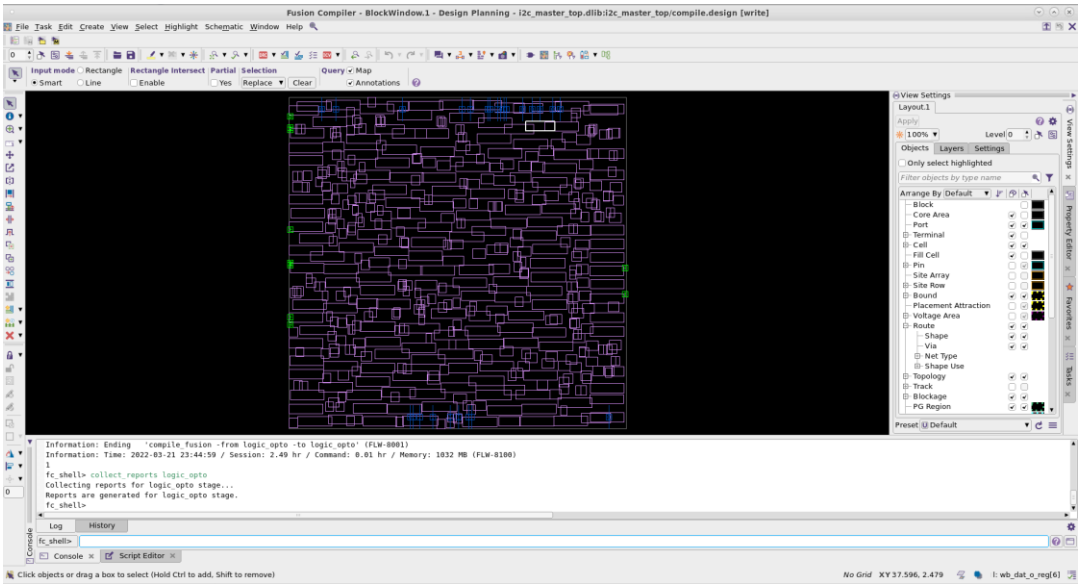

Fig. Layout View after logic_opto stage

To implement standard flow use *logic_opto_mcmm_**basic**.tcl* script. To check the design quality use the following report command to support scenarios created in MCMM:

```
redirect -file \
../../reports/${report_prefix}_report_power.rpt
{report_power -scenarios [all_scenarios]}
redirect -file \
../../reports/${report_prefix}_report_timing_setup.rpt
{report_timing -scenarios [all_scenarios] \
-delay_type max}
redirect -file \
../../reports/${report_prefix}_report_timing_hold.rpt
{report_timing -scenarios [all_scenarios] \
-delay_type min}
```

```
redirect -file \
../../reports/${report_prefix}_report_area.rpt \
{report_area}
```

Here, `${report_prefix}` variable is set to "logic_opto_**general**" to save the reports for comparison.

Controlling timing/power/area efforts

To focus on **timing** optimization use the following option before compiler_fusion command:

```
set_app_options -name compile.flow.high_effort_timing \
 -value 2
```

Values can be “0”,”1” and “2”. Value “2” is setting up most resource-consuming timing optimization. As design used for these labs is small, “2“ is used as an example.

To implement synthesis with focus on timing optimization use *logic_opto_mcmm_**effort_for_timing**.tcl* script. Here, `${report_prefix}` variable is set to "logic_opto_**timing**" to save the reports for comparison.

To focus on **area** optimization use the following option before compiler_fusion command:

```
set_app_options -name compile.flow.high_effort_area \
-value true
```

Value “true” is setting up area optimization.

To implement synthesis with focus on area optimization use *logic_opto_mcmm_**effort_for_area**.tcl* script. Here, `${report_prefix}` variable is set to "logic_opto_**area**" to save the reports for comparison.

To focus on **power** optimization use the following option before compiler_fusion command:

```
set_app_options -name compile.flow.enable_power \
-value true
```

Value “true” is setting up area optimization.

To implement synthesis with focus on area optimization use *logic_opto_mcmm_**effort_for_area**.tcl* script. Here, `${report_prefix}` variable is set to "logic_opto_**power**" to save the reports for comparison.

Now generated reports can be compared for timing/power/area values. Please note, in some cases the difference between values will not be noticable. It is due to the small size of the design used for this course.

All scripts are ending with the following commands to save corresponding blocks:

```
save_block
save_lib
close_block
close_lib
```