OASIS 3D-Router Hardware Physical Design

Technical Report

© Adaptive Systems Laboratory
Division of Computer Engineering
School of Computer Science and Engineering
University of Aizu

Contact: [d8141104, benab] @ u-aizu.ac.jp

1. **Synthesis**
2. **Place & Route**
3. **Design Checking: LVS (Layout-Versus-Schematic) and DRC (Design-Rule Check)**
4. **Post Layout Simulation**
5. **Pad Insertion**
6. **Acknowledgement**
1. Synthesis
Objectives

• After completing this tutorial you will be able to:
  – Synthesize 3D-OASIS-NoC (3D-ONOc) router using Design-Compiler CAD tool.
  – Place & Route (P&R) 3D-ONOc router with Cadence SoC-Encounter
  – Evaluate the area and power
  – Learn how to make the synthesis and P&R via:
    • The CAD Graphic User Interface
    • Tcl script
Contents

• Tutorial directory structure
• 3D-OASIS-NoC router hierarchy
• Environment
• Phase1:
  – Design Compiler Synthesis steps (Step 1~7)
• Phase2:
  – SoC Encounter Place & route steps (Step1~12)
• Script
Tutorial directory structure

Path: ~ /3D-NoC

- P&R
  - checkpoints
  - input_files
  - reports
  - script

- Synthesis
  - checkpoints
  - output_files
  - reports
  - script
  - verilog_src

- Readme.txt
You can check the complete tutorial’s directory structure by typing:

```
tree -d 3D_ONoC
```
under your home directory “~”
3D-NoC router hierarchy

Top level module

- `router_LAXYZ.v`
- `input_port.v`
- `sw_alloc.v`
- `crossbar.v`
- `fifo.v`
- `request_cntrl.v`
- `input_port.v`
- `matrix_arb_for_multistage.v`
- `stop_go.v`
- `mux.v`
Phase 1: Synthesis with Synopsis Design Compiler
Requirements

• In this first phase, we synthesize 3D-ONoC router and evaluate its area and power
• For this phase, we need the Verilog source files which are located in:
  ~/3D-ONoC/Synthesis/verilog_src
• We also need the .db, .sdb, and .slpdb library files which are located in: ~/lib
Synthesis directory structure

Path: ~/3D-NoC

- **Synthesis**
  - checkpoints
    - Contains the checkpoints saved all along the tutorial
  - output_files
    - Contains the files necessary for the Place and Route phase (.V and .sdc)
  - script
    - Contains the syn_LAXYZ.tcl shell script file
  - verilog_src
    - Contains the Verilog-HDL source files for the 3D-OASIS-NOC
- **P&R**
  - Contains the files necessary for the Place and Route phase (.V and .sdc)

- **Readme.txt**
  - Contains the reports generated from the compilation (Area, power)

Contains the reports generated from the compilation (Area, power)
You can check the complete Synthesis directory and file structure by typing:

tree Synthesis under the “3D_ONoC” directory
Environment

Initially *bash* will start, so type "**tcsh**" to start *csh*. 
Environment

Go to /home/zxp035/3D-ONoC/Synthesis where the Synthesis folder is located.
Environment

Type `design_vision` to start Design Compiler
Design Compiler: Synthesis steps

Welcome screen
Step 1: Library setup

First, we should specify the library for the design. Click **file-> Setup**
Step 1: Library setup
a- Target library

Select the **Target library**
Step 1: Library setup

a- Target library

Click on `you_library.db` to modify the default library
Step 1: Library setup

a- Target library

Go to ~/lib and select *typical.db*. Click *Open*
Step 1: Library setup
b- Symbol library

Select the Symbol library
Step 1: Library setup

b- Symbol library

Go to ~/lib and select **generic.sdb**.
Click **Open**
Step 1: Library setup

c- Synthetic library

Select the Synthetic library
Step 1: Library setup

c- Synthetic library

Go to ~/lib and select `dw_foundation.sldb`
Click Open
Step 1: Library setup

d- Link library

Finally the **Link library**, should contain the combined path of:
- Current directory: “*”
- Target library: “/home/zxp035/lib/typical.db”
- Synthetic library: “/home/zxp035/lib/dw.Foundation.sldb”

Click **OK**
Step 2: Analysis
a- File selection

After setting the libraries, we should make the analysis of our circuit. Click on File->Analyze.
Step 2: Analysis

a- File selection

1- Change the format to **Verilog**
2- Click **Add**
Step 2: Analysis

a- File selection

Go to ./verilog_src and select the 11 Verilog files of the router

Click Select
Step 2: Analysis

a- File selection

Click OK
Step 2: Analysis
b- Analysis report

The analysis report will appear in the console window.
Step 2: Analysis c- Checkpoint

1. Click File-> Save As
2. Go to ./checkpoints
3. In File name, type analysis.ddc
4. Change the Format to DDC (ddc)
5. Click Save

It is recommended to save your progress after each step
Step 3: Elaboration

a- Top file selection

After Analysis, we should make the Elaboration of our circuit. Click on File->Elaborate in Design, select the top module of the router `router_LAXYZ (verilog)`
Step 3: Elaboration
a- Top file selection

Click OK
Step 3: Elaboration

b- Elaboration report

The elaboration report will appear in the console window. Ignore the warnings.
Step 3: Elaboration

c- Hierarchy

The design hierarchy can be seen on the left side of the window.
Step 3: Elaboration
d- Schematic

The design schematic can be seen by clicking on this icon at the top of the window.
Step 3: Elaboration e- Checkpoint

1. Click File-> Save As
2. Go to ./checkpoints
3. In File name, type elaboration.ddc
4. Make sure the Format is DDC (ddc)
5. Click Save
Step 4: Constraints setting

a- Clock

1. Set clock name: clk
2. Set Period: 10.0
3. Set Rising: 0.00
4. Set Falling: 5.00
5. Check Don’t touch network

Click OK

Click Attributes -> Specify Clock
Step 4: Constraints setting
b- Wire load

- Click Attributes -> Operating Environment-> Wire Load…
- Select 5K_hvratio_1_1 and then click OK
Step 5: Compilation

1. Set Map effort to **high**
2. Check **Incremental mapping**

Click **OK**

- Click **Design-> Compile Design**
Step 6: Report a- Area

Click on **Design-> Report Area** then **OK**.
In this library the wire load models do not include area information (Wire load has zero net area) so the Net Interconnect area (and therefore Total area) is left undefined. The area is measured in micrometer (um)
Step 6: Report b- Power

Click on Design-> Report Power then OK.
Step 6: Report b- Power

The report shown above should appear giving information about Cell Internal Power, Net Switching Power, Total Dynamic Power, and the Cell Leakage Power, in addition to their percentage from the total power consumption.
Step 7: Output files  
a- Verilog file

1. Click File-> Save As  
2. Go to ./output_files  
3. In File name, type router_LAXYZ.vnet  
4. Change the Format to Verilog (V)  
5. Click Save

After finishing the compilation, we should generate the necessary files for the next Place&Route step
Step 7: Output files

b- sdc file

In the dc_shell window type the following command to save the .sdc file

```
write_sdc ./output_files/router_LAXYZ.sdc
```
Step 7: Output files
e- checkpoint

1. Click **File-> Save As**
2. Go to **./checkpoints**
3. In **File name**, type **router_LAXYZ.ddc**
4. Make sure the **Format** is **DDC (ddc)**
5. Click **Save**
Scripts

• The 7 steps previously presented can be made via commands inserted in the dc_shell.
• The commands required for the synthesis are grouped in a single .tcl file.
• The .tcl file is named “syn_LAXYZ.tcl”
• It is located in: “/home/zxp035/3D_ONoC/Synthesis/script"
To run the TCL script, click **File→ Execute script**

Go to **./scripts**, select **syn_LAXYZ.tcl** and click **Open**
#### Define the variable which we will use ####

```tcl
set base_name "router_LAXYZ"
set clock_name "clk"
set clock_period 10.0
```

#### Step 1: Set the libraries: ####

```tcl
set target_library "~/lib/typical.db"
set synthetic_library "~/lib/dw_foundation.sldb"
set link_library [concat "*" $target_library $synthetic_library]
set symbol_library "~/lib/generic.sdb"
define_design_lib WORK -path ./WORK # redirect the log files to a new folder “WORK”
```

#### Step 2: Analysis ####

```tcl
analyze -format verilog {./verilog_src/crossbar.v ./verilog_src/defines.v ./verilog_src/fifo.v ./verilog_src/input_port.v ./verilog_src/matrix_arb_formultistage.v ./verilog_src/mux_out.v ./verilog_src/request_cntrl.v ./verilog_src/route.v ./verilog_src/router_LAXYZ.v ./verilog_src/stop_go.v ./verilog_src/sw_alloc.v}
```
# Analysis checkpoint
write_file -format ddc -hierarchy -output ./checkpoints/analysis.ddc

#### Step 3: Elaboration####
elaborate $base_name

# Elaboration checkpoint
write_file -format ddc -hierarchy -output ./checkpoints/elaboration.ddc

#### Step 4: Constraints####
# Clock
create_clock -name $clock_name -period $clock_period [find port $clock_name]
set_clock_uncertainty 0.02 [get_clocks $clock_name]
# Delay
set_input_delay 0.1 -clock clk [remove_from_collection [all_inputs] {clk reset}]
set_output_delay 0.1 -clock clk [all_outputs]
# Wire load
set_wire_load_model -name 5K_hvratio_1_1 -library NangateOpenCellLibrary
#### Step 5: Compilation####

```
compile -map_effort high
compile -incremental_mapping -map_effort high
```

#### Step 6: Report####

```
# Summary report to be saved under the “reports” folder
report_qor > ./reports/Summary_report_${base_name}.txt
# Hierarchical area report to be saved under the “reports” folder
report_area -hierarchy > ./reports/report_area_${base_name}.txt
```

#### Step 7: Output files ####

```
# verilog file
write -format verilog -hierarchy -output ./output_files/${base_name}.vnet
# sdc file
write_sdc ./output_files/${base_name}.sdc
# Final checkpoint
write_file -format ddc -hierarchy -output ./DB/${base_name}.ddc
```
2. Place & Route
Requirements

• After we finished the synthesis phase, we proceed to perform the Place and Route of 3D-ONoC router with Cadence SoC Encounter.

• For this phase, we need the .vnet and .sdc files obtained from the synthesise phase and which are located in:
  ~/3D-ONoC/Synthesis/output_files

• We also need the .lib and .lef library files which are located in: ~/lib
Place and Route directory structure

Path: ~/3D-NoC

- **Synthesis**
  - **checkpoints**: Contains the checkpoints saved all along P&R tutorial

- **P&R**
  - **input_files**: Contains the files necessary files for the Place and Route phase (.V and .sdc)

- **Readme.txt**
  - **script**: Contains the par_LAXYZ.tcl shell script file

- **reports**: Contains the reports generated from the post P&R compilation
You can check the complete Synthesis directory and file structure by typing:

```
tree Synthesis
```
under the “3D_ONoC” directory
Environment

Make sure that you are working under **cshr** environment. Otherwise type **tcsh**. Go to **/home/zxp035/3D-ONO\_C/P\&R** where the P&R folder is located.
Environment

First, we need to copy the `router_LAXYZ.vnet` and `router_LAXYZ.sdc` files generated from the synthesis phase which will be used as input for the P&R phase. Type:

```
% cp ../Synthesis/output_files/router_LAXYZ.vnet ./input_files/
% cp ../Synthesis/output_files/router_LAXYZ.sdc ./input_files/
```
Environment

Type **velocity** to start SoC Encounter
SoC Encounter: P&R steps

Welcome screen
Step 1: Import Design

Click on File->Import Design
Click Files to import the netlist
Step 1: Import Design

a- Netlist (.vnet)

1. Click on >> to expand
2. Go to ./input_files folder
3. Double click on router_LAXYZ.vnet
4. Click Close
Step 1: Import Design

b- Top module

1. In Top Cell: type `router_LAXYZ`
2. Click on LEF files
Step 1: Import Design

c- LEF file

1. Click on >> to expand
2. Go to ~/lib folder
3. Double click on NangateOpenCellLibrary.lef
4. Click Close
Step 1: Import Design

1. Click on **Advanced**
2. In **IPO/CTS** type **CLKBUF_X1 CLKBUF_X2 CLKBUF_X3**
Step 1: Import Design

d- Advanced settings

1. In Power type:
   a. VDD in Power nets
   b. VSS in Ground Nets

2. Click back on Basic
   (DO NOT click on OK)
Step 1: Import Design

e- Analysis Configuration

Click on **Create Analysis Configuration**
Step 1: Import Design

e-Analysis Configuration

1. Double click on **Library Sets** in the **MMMC Browser** window
2. On the add **Library Set** window, type **default** in **Name**
3. Click on **Add**.
4. In the **Timing Library** Window, go to ~/lib and select **typical.lib**
5. Click **Open** (Timing Library Window) and then **OK** (Library Set Window)
Step 1: Import Design

e-Analysis Configuration

1. Double click on **Delay Corners** in the MMMC browser window
2. On the **Add Delay Corner** window, type **default** in **Name**
3. Change the **Library Set** to **default**.
4. Click **OK**
Step 1: Import Design
e- Analysis Configuration

1. Double click on **Constraint Modes** in the MMMC browser window.
2. On the Add Constraint Mode window, type **default** in Name.
3. Click on Add.
4. In the SDC Constraint File window, go to ./input_files and select **router_LAXYZ.sdc**.
5. Click **Open** (SDC Constraint File window) and then **OK** (Add Constraint Mode window).
1. Double click on **Analysis Views** in the **MMMC browser** window
2. On the **Add Analysis View** window, type **default** in Name
3. Click **OK**
Step 1: Import Design

e-Analysis Configuration

1. Double click on **Setup Analysis Views** in the **MMMC browser** window
2. In the **Add Setup Analysis View** window, make sure that Analysis View is set to **default**
3. Click **OK**
Step 1: Import Design

Analysis Configuration

1. Double click on **Hold Analysis Views** in the **MMMC browser** window
2. In the **Add Hold Analysis View** Window, make sure that **Analysis View** is set to **default**
3. Click **OK**
Step 1: Import Design

e- Analysis Configuration

1. Click on Save&Close… in the MMMC browser window
2. Go to ./input_files
3. Type Default.view in File name
4. Click Save
Step 1: Import Design

Result

In the welcome screen, we can see the modules of 3D-ONoC router before placement: 7 input_ports: (ip[0~6]), Switch_allocator (sw_allc), and Crossbar (cbar)
Step 1: Import Design

- **g- Checkpoint**

  We should save the progress at each step.

  **Click File-> Save Design**

1. Click in *File name*
2. Go to */checkpoints*
3. Type *import.enc* in *File name*
4. Click *Save*
5. Click *OK*
Step 2: Floorplan

1. Check **Die Size by:**
2. Enter **300** for both **Width** and **Height**
3. Enter **15** for
   - **Core to Left**
   - **Core to Right**
   - **Core to Top**
   - **Core to Bottom**
4. Click **OK**

In this step we specify the floorplan
Click **Floorplan-> Specify Floorplan**
Step 2: Floorplan

1. In the main window, the boundaries of the chip appear.
2. Save your design under `floorplan.enc` in `./checkpoints` directory.
Step 3: Power Ring

1. Input **VDD VSS** in *Nets(s):*
2. Change the layer to:
   - **metal 10 V** for *Top* and *Bottom*
   - **metal 9 H** for *Left* and *Right*
3. Change the *Width* to **4**
4. Change the *Spacing* to **2**
5. Check **Center in channel**
6. Click **OK**
Step 3: Power Ring

In the main window, the power ring appears.
Step 4: Power Stripe

1. Input **VDD VSS** in Nets(s):  
2. Change Layer to **metal 8**  
3. Set:  
   - Width to **4**  
   - Spacing to **2**  
4. Change **Set-to-set- distance** to **50**  
5. Change the **Relative from core or selected area**: X from Left to **35**  
6. Click **OK**

Click on **Power-> Power Planning->Add Stripe** …
Step 4: Power Stripe

In the main window, the power stripes appear.
Step 5: Power Routing

- Click on **Route** > **Special Route**
- Input **VDD VSS** in *Nets(s):*
- Click **OK**
Step 5: Power Routing

1. In the main window, the power routing appears.
2. Save your design under `power.enc` in `.checkpoints` directory.
Now we place the 3D_ONoC modules on the die:
- Click **Place**-> **Place Standard Cell**.
- Click **OK**
Step 6: Placement

In the main window, click on to view the placed modules.
Step 6: Placement

In the main window, click on for the physical view.
Step 7: Clock Tree

a- Synthesize

1. Click on Gen Spec
2. Select CLKBUF_X1 CLKBUF_X2 CLKBUF_X3
3. Click OK (Generate clock spec window)
4. Click OK (Synthesize Clock Tree)
Step 7: Clock Tree b- Display

1. Check **Clock Route Only**
2. Check **Display Clock Tree**
3. Click **OK**

Click on **Clock -> Display -> Display Clock Tree**
Step 7: Clock Tree

1. In the main window, the clock tree appears
2. Save your design under `clock_syn.enc` in `.checkpoints` directory
Step 8: Nano Route a- Setting

Click on Route-> NanoRoute-> Route

Check Time Driven and then click OK
Above is a sample of the report generated after the Nano Route step. It gives information about the wire length, metal used, etc.
Step 9: Optimization a- setting

1. Check **Post-Route**
2. Check **Hold**
3. Click **OK**

Click on **Optimize-> Optimize Design**
Step 9: Optimization

b- report

Above is a sample of the report generated after the Optimization step. It gives information about the Setup and Hold violations, used metal layers thickness, etc.
Step 10: Adding Fillers

1. Check Select
2. Select FILLCELL_X1,2,4,8,16,32
3. Click ADD
4. Click Close
5. Click OK

Click on Place -> Physical Cells->Add Filler
Step 10: Adding Fillers

1. Save your final design under `final.enc` in `.\checkpoints` directory
Step 11: Design checking

a- Layout Vs. Schematic (LVS)

Click **Verify->Verify Connectivity**

Save the **Verify Connectivity Report.rpt** under **./reports**, and then click **OK**

---

Report displayed on the terminal

```
****** Start: VERIFY CONNECTIVITY ******

Design Name: router_LAXYZ
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (250.0000, 250.0000)
Error Limit = 1000; Warning Limit = 50
Check all nets
**** 20:14:24 **** Processed 5000 nets (Total 6074)
Time Elapsed: 0:00:01.0

Begin Summary
Found no problems or warnings.
End Summary

****** End: VERIFY CONNECTIVITY ******
Verification Complete : 0 Viols. 0 Wrngs.
(CPU Time: 0:00:00.2 MEM: 0.004M)
```
Step 11: Design checking

b- Design Rule Check (DRC)

Click Verify -> Verify geometry

Under Advanced, save the Verify Geometry Report.rpt under ./reports.

Click OK

Report displayed on the terminal
Step 12: Output files

a- SPEF file

1. Click on **Timing-> Extract RC**
2. Check **Save SPEF to**
3. Click **OK**

We save the output files of the Place&Route phase under ./*reports*/
Step 12: Output files
b- SDF file

1. Click on **Timing-> Extract RC**
2. Save the **SDF Output File** in **./reports**
3. Click **OK**

We save the output files of the Place&Route phase under **./reports**
Step 12: Output files
c- Netlist file

1. Click on File-> Save-> Netlist
2. Save the `router_LAXYZ_final.vnet` in `.reports`
3. Click OK

We save the output files of the Place&Route phase under `.reports`
The 12 steps previously presented can be made via commands inserted in the SoC Encounter terminal.

The commands required for the Place&Route are grouped in a single `.tcl` file.

The `.tcl` file is named “`par_LAXYZ.tcl`”

It is located in:

`/home/zxp035/3D-ONoC/P&R/script`

To run the TCL script type on your terminal:

```
velocity 1> source /3D-NoC/P&R/script/par_LAXYZ.tcl
```
# Step 1: Setup (File --> Import Design)

setUIVar rda_Input ui_netlist .input_files/router_LAXYZ.vnet
setUIVar rda_Input ui_timingcon_file .input_files/router_LAXYZ.sdc
setUIVar rda_Input ui_topcell router_LAXYZ
setUIVar rda_Input ui_leffile ~/lib/NangateOpenCellLibrary.lef
setUIVar rda_Input ui_timelib ~/lib/typical.lib
setUIVar rda_Input ui_pwrnet VDD
setUIVar rda_Input ui_gndnet VSS
setUIVar rda_Input ui_cts_cell_list {CLKBUF_X1 CLKBUF_X2 CLKBUF_X3}
commitConfig

# Checkpoint
saveDesign import.enc
Script: par_LAFT.tcl (2/8)

# Step 2: Floorplan (Floorplan --> Specify Floorplan)
#
floorPlan -s 300 300 15 15 15 15
# Checkpoint
saveDesign floor.enc

#
# Step 3: Power ring (Power --> Power Planning --> Add Ring)
#
createPGPin VDD -net VDD
createPGPin VSS -net VSS
globalNetConnect VDD -type pgpin -pin VDD -sinst dp/rf
globalNetConnect VSS -type pgpin -pin VSS -sinst dp/rf
addRing -nets {VSS VDD} -type core_rings \ 
  -spacing_top 2 -spacing_bottom 2 -spacing_right 2 -spacing_left 2 \ 
  -width_top 4 -width_bottom 4 -width_right 4 -width_left 4 \ 
  -around core -jog_distance 0.095 -threshold 0.095 \ 
  -layer_top metal10 -layer_bottom metal10 -layer_right metal9 \ 
  -layer_left metal9 \ 
  -stacked_via_top_layer metal10 -stacked_via_bottom_layer metal1

#

# Step 4: Power stripe (Power --> Power Planning --> Add Stripe)
#

addStripe -nets {VSS VDD} -layer metal8 -width 2 -spacing 1.5 \ 
  -block_ring_top_layer_limit metal9 -block_ring_bottom_layer_limit metal7 \ 
  -padcore_ring_top_layer_limit metal9 -padcore_ring_bottom_layer_limit metal7 \ 
  -stacked_via_top_layer metal10 -stacked_via_bottom_layer metal1 \ 
  -set_to_set_distance 50 -xleft_offset 50 -merge_stripes_value 0.095 \ 
  -max_same_layer_jog_length 1.6
Script: par_LAFT.tcl (4/8)

# Step 5: Power route (Route --> Special Router)
#

sroute -nets {VSS VDD} -layerChangeRange {1 10} \    
   -connect { blockPin padPin padRing corePin floatingStripe } \    
   -blockPinTarget { nearestRingStripe nearestTarget } \    
   -padPinPortConnect { allPort oneGeom } \    
   -checkAlignedSecondaryPin 1 -blockPin useLef -allowJogging 1 \    
   -crossoverViaBottomLayer 1 -allowLayerChange 1 -targetViaTopLayer 10 \    
   -crossoverViaTopLayer 10 -targetViaBottomLayer 1

# Checkpoint
saveDesign power.enc
#

# Step 6: Placement (Place --> Standard Cell)
#

placeDesign -prePlaceOpt
# Step 7: Clock tree synthesis (CTS)
#a- Synthesis: (Clock --> Synthesize Clock Tree)

addCTSCellList {CLKBUF_X1 CLKBUF_X2 CLKBUF_X3}
clockDesign -genSpecOnly Clock.ctstch
clockDesign -specFile Clock.ctstch -outDir clock_report -fixedInstBeforeCTS

# Checkpoint
saveDesign clock_syn.enc

#
# b- Display: (Clock --> Display --> Display Clock Tree)
This step should be done manually
#
# Step 8: Detailed route (Route --> Nano Route --> Route)

#
setNanoRouteMode -quiet -routeWithTimingDriven true
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven true
routeDesign -globalDetail

#
# Step 9: Optimization (postRoute) (Optimize --> Optimize Design)
#
optDesign -postRoute
optDesign -postRoute -hold

# Checkpoint
saveDesign route.enc
# Step 10: Add fillers (Place --> Physical Cells --> Add Filler)
addFiller -prefix FILLER -cell FILLCELL_X1 FILLCELL_X2 FILLCELL_X4 \  FILLCELL_X8 FILLCELL_X16 FILLCELL_X32

# Step 11: Verification (LVS) (Verify --> Verify Connectivity)
verifyConnectivity -type all -error 1000 -warning 50 # LVS check
verifyGeometry # DRC

# Step 12: Data out (Timing --> Extract RC, Timing --> Write SDF, File --> Save --> Netlist)
saveNetlist router_LAXYZ_final.vnet # Netlist
isExtractRCModeSignoff
rcOut -spef router_LAXYZ.spef # SPEF file
delayCal -sdf router_LAXYZ.sdf –idealclock # SDF file

# Final checkpoint
save Design final.enc

##############################################################################
################################### End tutorial #################################
##############################################################################
3. Design checking  LVS (Layout- Versus- Schematic) & Design Rule Check (DRC)
Objectives

• In this tutorial, we check the correctness of the designed 3D-ONoC router. Two main checking processes are performed in this tutorial:
  
  – **Layout Versus Schematic (LVS)**
    
    - Checks whether the integrated circuit layout in the Place & Route phase (Phase 2) corresponds to the original schematic or circuit diagram of the design obtained in the Design Synthesis phase (Phase 1).
  
  – **Design Rule Check (DRC)**
    
    - Ensures that the layout conforms to the rules designed/required for faultless fabrication.
Requirements

• Before starting the design check, you should have already finished the two previous steps:
  – Design synthesis (DS)
  – Place & Route (P&R)

• If you are not continuing the previous two steps, you need the `final.enc` file and `file.enc.dat` folder to be copied first to the `./checkpoints` directory to restore the final post P&R design
Design Check directory structure

Path: ~/3D-NoC

- Synthesis
- P&R
  - Design_check
    - checkpoint
      Contains the last checkpoint saved from P&R tutorial. It will be used to restore the design and perform the checking
- Readme.txt
Contents

- Environment
- Restore design
- Layout Vs. Schematic (LVS)
- Design Rule Check (DRC)
- Commands
- Make sure that you are working under `cshr` environment. Otherwise type `tcsh`.
- Go to `/home/zxp035/3D-ONoC/` and make `Design_check`
- In the new `Design_check` directory, make a new directory `checkpoint`
In the new **checkpoint** directory, we will copy the last checkpoint performed in P&R phase. We need to copy `final.enc` file and `final.enc.dat` folder.
Use `cp` command to copy `final.enc` file from `../PandR/checkpoints` into `./checkpoint`.

Use `cp -r` command to copy also `final.enc.dat` folder.
Environment

```
$ tcsh
/home/zxp035% cd 3D_ONoC/
/home/zxp035/3D_ONoC% mkdir Design_check
/home/zxp035/3D_ONoC% cd Design_check/
/home/zxp035/3D_ONoC/Design_check% mkdir checkpoint
/home/zxp035/3D_ONoC/Design_check% cp .../PandR/checkpoints/final.enc ./checkpoint/
/home/zxp035/3D_ONoC/Design_check% cp -r .../PandR/checkpoints/final.enc.dat ./checkpoint/
/home/zxp035/3D_ONoC/Design_check% velocity
```

Start SoC Encounter
First, we should restore the final design of the P&R phase. Click File>Restore design.

1. Click on the folder.
2. Go to `./checkpoint`.
4. Click **Open**.
5. Click **OK**.
The final layout of the P&R phase should appear
Layout Vs. Schematic (LVS)

Report displayed on the terminal

Click **Verify->Verify Connectivity**

Save the **Verify Connectivity Report.rpt** under **./reports**, and then click **OK**
Design Rule Check (DRC)

Under Advanced, save the Verify Geometry Report.rpt under ./reports.

Click OK
Commands

• To perform the Layout Vs. Schematic (LVS) type the following command on your terminal:

```
velocity 1> verifyConnectivity -type all -error 1000 -warning 50
```

• To perform the Design Rule Check (DRC) type the following command on your terminal:

```
velocity 2> verifyGeometry
```
4. Post-Layout simulation
Objectives

After completing this tutorial you will be able to:

- Check if the post-layout design is free from any timing violations
- Report timing and area
- Evaluate the power consumption (dynamic and static)
- Learn how to make the post-layout simulation via:
  - The CAD Graphic User Interface
  - Tcl script
Contents

• Requirements
• Post-layout simulation directory structure
• Setup
• Post layout synthesis (Step 1~3)
• Script
Requirements

- Before starting the post-layout, you should have already finished the three previous phases:
  - Design Synthesis (DS)
  - Place & Route (P&R)
  - Design Check (LVS and DCR)

- We should create a new directory: 
  ~/3D-ONoC/Post
  where the post-layout simulation is performed.
Post-layout directory structure

Path: ~/3D-NoC

- **Synthesis**
  - **verilog_src**
    - Contains the Verilog-HDL source files for the 3D-OASIS-NOC

- **P&R**
  - **input_files**
    - Contains the files necessary for the Post-layout phase (.Vnet, .sdc, .spef, .sdf, and .sdc)

- **Design_check**
  - **scripts**
    - Contains the sta_LAXYZ.tcl and power_LAXYZ.tcl shell script files

- **Post**
  - **reports**
    - Contains the reports generated from the post-layout synthesis compilation

- **Readme.txt**
Setup

• Before we start, we should copy some output files that we will use for this phase.
  – router_LAXYZ_final.vnet  (From P&R phase)
  – router_LAXYZ_final.spef  (From P&R phase)
  – router_LAXYZ_final.sdf   (From P&R phase)
  – router_LAXYZ_final.sdc   (From Synthesis phase)

• These files should be copied to
  ~/3D-ONoC/Post/input
Type `tcsh` and go to `~3D-ONoC/Post`

Using the "cp" command, copy the necessary four files to `./input`

Start Design Compiler by typing `design_vision`
Step 1: Timing analysis

• For this step, we execute a script that contains the necessary operations for time analysis.

• The operations are almost the same as the ones performed in Phase 2 (Design Synthesis) of this tutorial.

• The \texttt{sta\_LAXYZ.tcl} script file needed for this step is located in \texttt{./scripts}.

• Next slides depicts \texttt{sta\_LAXYZ.tcl}
Step 1: Timing analysis: sta_LAXYZ.tcl (1/3)

#### Define the variable which we will use ####

```tcl
set base_name    "router_LAXYZ"
set vnet_file    "router_LAXYZ_final.vnet"
set spef_file    "router_LAXYZ.spef"
set sdf_file     "router_LAXYZ.sdf"
set sdc_file     "router_LAXYZ.sdc"
```

#### Step 1: Set the libraries: ####

```tcl
set target_library "~/lib/typical.db"
set synthetic_library "~/lib/dw_foundation.sldb"
set link_library [concat "*" $target_library $synthetic_library]
set symbol_library ""~/lib/generic.sdb"
define_design_lib WORK -path ./WORK # redirect the log files to a new folder "WORK"
```
### Step 2: Read post_layout netlist###
read_file -format verilog ./input/$vnet_file
current_design $base_name
link

### Delay and RC information###
read_sdc ./input/$sdc_file
read_sdf ./input/$sdf_file
read_parasitics ./input/$spef_file

### Generate reports###
report_timing > ./reports/timing_report_${base_name}.txt
report_reference -hier > ./reports/reference_report_${base_name}.txt
Step 1: Timing analysis
Execute the script

To run the TCL script, click **File** > **Execute script**
Go to ./scripts, select **sta_LAXYZ.tcl** and click **Open**
Step 1: Timing analysis Reports

Analysis reports for timing and area will be saved in `./reports`. The reports contain detailed evaluation of reference (area) and timing delay by module.
Step 2: Timing simulation

• In this second step, we will check whether our design is free from any delay violation.
• We will use `ncverilog` and `simvision`.
• We created a test bench file, named `Test.v`, in order to evaluate 3D-ONoC router.
• In this test bench, random flits are generated, injected from the 7 input-ports of the router and ejected from the 7 output ports.
• Finally, the correctness of the ejected flits is checked.
Step 2: Timing simulation: a- Hierarchy

Test.v is located in ./verilog_src along with the remaining Verilog source files
Step 2: Timing simulation: 

\textit{b-Test.v} (1/4)

\begin{verbatim}
`timescale 1ns/1ns

module Test;
    reg               clk;
    reg               reset;

//wire list object value output variables
wire [37:0]       dat_out_local, dat_out_north, dat_out_east, dat_out_south, dat_out_west, dat_out_up, dat_out_down;
wire [6:0]        stop_out;

//register list object value of input test
reg [37:0]        dat_in_local, dat_in_north, dat_in_east, dat_in_south, dat_in_west, dat_in_up, dat_in_down;

//******* registers used for the payload of input data *****\
    reg [20:0] payload_1;
    reg [20:0] payload_2;
    reg [20:0] payload_3;
    reg [20:0] payload_4;
    reg [20:0] payload_5;
    reg [20:0] payload_6;
    reg [20:0] payload_7;

......
\end{verbatim}
Step 2: Timing simulation:

b- Test.v (2/4)

```verilog
//Top module definition
router_LAXYZ router (.clk(clk),
    .reset(reset),
    .data_in({dat_in_down, dat_in_up, dat_in_west, dat_in_south, dat_in_east, dat_in_north, dat_in_local}),
    .data_out({dat_out_down, dat_out_up, dat_out_west, dat_out_south, dat_out_east, dat_out_north, dat_out_local}),
    .stop_in({stp_in_local, stp_in_norh, stp_in_east, stp_in_south, stp_in_west, stp_in_up, stp_in_down}),
    .stop_in(7'd0000000),
    .stop_out(stop_out),
    .xaddr(3'b010), .yaddr(3'b010), .zaddr(3'b010)); // We assume that the router has 222 address

//clock generation (100 Mhz frequency )
always #5000 clk = ~clk;

//Annotation file initialization
initial begin
`ifdef __POST_PR__
    `ifdef $sdf_annotate("input/router_LAXYZ.sdf", Test.router, "sdf.log", "MAXIMUM")
`endif

#0
clk = 1;
reset = 1'b1;
```

```
```
#100000

//Initialization of the vcd file that collects simulation information
$dumpfile("dump.vcd");
$dumpvars(0, Test);

//Start sending flits
for(i=0;i<100;i=i+1)begin  //We assume the number of sent flit is 100 for simplicity
  #10000
  //*** local port sending
  if(stop_out[0] == 1)
    dat_in_local = 0;
  else begin
    dat_in_local = {payload_1,9'b010011010,7'b0000010,1'b1};//(0,1)
    payload_1 = payload_1 + 1;
    sent1 = sent1 + 1;
  end

  //*** We perform the same operations for the remaining input-ports
Step 2: Timing simulation: b-Test.v (4/4)

```verilog
end // for loop end

#100000
$finish;

end // initial begin
always @(dat_out_local) begin // Count the flit received at the local out-port
rec1 = rec1 + 1;
end
rec5 = rec5 + 1;
end

always @(dat_out_north) begin // Count the flit received at the north out-port
rec2 = rec2 + 1;
end

..... // Count the flit received at the remaining out-ports
always @(dat_out_down) begin
rec7 = rec7 + 1;
end
endmodule // The end of Test.v
Step 2: Timing simulation: c- Compilation

- Using `ncverilog` we compile our test bench `Test.v` with our top module netlist file `router_LAXYZ_final.vnet`.
- The result of this compilation is the `dump.vcd` file previously initialized in `Test.v`.
- In your terminal and under `~/3D-ONoC/Post` type the following command:

  ```bash
  /home/zxp035/3D_ONoC/Post% ncverilog +access+r +define+__POST_PR__ verilog_src/Test.v input/router_LAXYZ_final.vnet -v ~/lib/typical.lib
  ```
Step 2: Timing simulation: d- Simulation

- Now, we launch `simvision` to see the result of the simulation
- In your terminal and under `~/3D-ONoC/Post` type the following command:

  ```bash
  /home/zxp035/3D_ONoC/Post% simvision &
  ```
Step 2: Timing simulation: d- Simulation

Simvision welcome screen
Click on File> Open Database ...
Step 2: Timing simulation:

d- Simulation

Change *Files of type* to VCD files (*.vcd)

Select **dump.vcd**

Click **Open**
Step 2: Timing simulation: d- Simulation

File Translation window will appear
Click on OK to translate dump.vcd into dump.trn
dump.trn file will be used to visualize the test bench signals
Click on **Test** to see the different variables used in the simulation

Click on **[visualize]** to visualize the signals
Step 2: Timing simulation:

d- Simulation

The waveform window appears

Click on to fit the window with your signals
Step 2: Timing simulation: d- Simulation

From the waveform we can see that:
- The 100 flits that we sent arrived to their destinations
- No time violations are found, otherwise the signals will be red instead of green
Step 3: Power evaluation

- After we made sure that there are no time violations in our design, we proceed to evaluate the power consumption.
- The `dump.vcd` file contains the switching activities information of the test bench. We need to convert the `.vcd` file into `.saif` file.
- The `.saif` file will be used by Design Compiler Power Analyzer to evaluate the power.
- In your terminal and under `~/3D-ONoC/Post` type the following command:

  ```sh
  /home/zxp035/3D_ONoC/Post% vcd2saif -input dump.vcd -output router_LAXYZ.saif
  ```
Step 3: Power evaluation

- For the evaluation, we execute a script that contains the necessary operations for power evaluation.
- The operations are almost the same as in the ones performed in Timing analysis (Step 1) of this post_layout simulation phase.
- The `power_LAXYZ.tcl` script file needed for this step is located in `./scripts`
- Next slides present `power_LAXYZ.tcl` file
Step 3: Power evaluation

(power_LAXYZ.tcl (1/2))

```tcl
#### Define the variable which we will use ####
set base_name  "router_LAXYZ"
set vnet_file  "router_LAXYZ_final.vnet"
set spef_file  "router_LAXYZ.spef"
set sdf_file  "router_LAXYZ.sdf"
set sdc_file  "router_LAXYZ.sdc"
set saif_file  "router_LAXYZ.saif"

#### Step 1: Set the libraries: ####
set target_library "~/lib/typical.db"
set synthetic_library "~/lib/dw.Foundation.sldb"
set link_library [concat "*" $target_library $synthetic_library]
set symbol_library """~/lib/generic.sdb"
define_design_lib WORK -path ./WORK # redirect the log files to a new folder “WORK”
```
### Step 2: Read post_layout netlist###
read_file -format verilog ./input/$vnet_file
current_design $base_name
link
### Delay and RC information###
read_sdc ./input/$sdc_file
read_sdf ./input/$sdf_file
read_parasitics ./input/$spef_file
### Read switching activities information###
reset_switching_activity
read_saif -input $saif_file -instance top/dut -unit ns -scale 1
### Generate reports###
report_timing > ./reports/timing_report_${base_name}.txt
report_reference -hier > ./reports/reference_report_${base_name}.txt
report_power -hier > ./reports/power_report_${base_name}.txt
Step 3: Power evaluation
Execute the script

To run the TCL script, click File> Execute script in Design Compiler
Go to ./scripts, select power_LAXYZ.tcl and click Open
Step 3: Power evaluation

Reports

A detailed power report is shown in dc_shell console. The report shows the static, dynamic and switching power consumption per module.

The total power consumption of the 3D-ONoC router is **222.387 uW**:  
- Leakage (static): 210 uW  
- Internal (dynamic): 7.763 uW  
- Switching (net): 4.395 uW
5. Pad Insertion
Objectives

After completing this tutorial you will be able to:

- Reperform the Place and Route phase while inserting the Input/Output (IO) pads
- Establish the connection between the IO pins/IO pads, and the input signals of 3D-ONoC router
- Generate the final netlist and other output files

This tutorial is performed is based on modifying input-files and creating a TCL script
Contents

• Requirements
• Pad Insertion directory structure
• Environment
• Step 1: Modify router_LAXYZ.vnet
• Step 2: Make router_LAXYZ.io
• Step 3: Make Iopad_LAXYZ.tcl script
• Step 4: Script execution
• Final layout
Requirements

• Before starting the post-layout, you should have already finished the four previous phases:
  – Design Synthesis (DS)
  – Place & Route (P&R)
  – Design Check (LVS and DCR)
  – Post-layout simulations

• We should create a new directory: 
  ~/3D-ONoC/lopad
  where the IO pad insertion is performed.
Post-layout directory structure

Path: ~/3D-NoC

- **Synthesis**
  - `input_files`
    - Contains the files necessary for the pad-insertion phase (.vnet, .sdc, and .io)

- **P&R**
  - `script`
    - Contains the ioapad.tcl shell script

- **Design_check**
  - `checkpoints`
    - Contains the checkpoints saved all along P&R tutorial

- **Post**
  - `output`
    - Contains the final output files generated from the IO pad insertion

- **Iopad**
  - `Readme.txt`

Environment

Make sure that you are working under **cshr** environment. Otherwise type **tcsh**.
Go to `/home/zxp035/3D-ONoC/lopad` where the Pad folder is located.

```bash
$ tcsh
/home/zxp035/3D-ONoC% cd Iopad/
/home/zxp035/3D-ONoC/Pad%
```
First, we need to copy the `router_LAXYZ.vnet` and `router_LAXYZ.sdc` files generated from the synthesis phase which will be used as input for the Pad insertion phase. Type:

```
% cp ../Synthesis/output_files/router_LAXYZ.vnet ./input
% cp ../Synthesis/output_files/router_LAXYZ.sdc ./input
```
module router_LAXYZ (  clk_pad, reset_pad, data_in_pad, data_out_pad, stop_in_pad, stop_out_pad,  
xaddr_pad,  
yaddr_pad, zaddr_pad );

 input [265:0] data_in_pad;  
output [265:0] data_out_pad;  
input [6:0] stop_in_pad;  
output [6:0] stop_out_pad;  
input [2:0] xaddr_pad;  
input [2:0] yaddr_pad;  
input [2:0] zaddr_pad;  
input clk_pad, reset_pad;

 wire [265:0] data_in;  
wire [265:0] data_out;  
wire [6:0] stop_in;  
wire [6:0] stop_out;  
wire [2:0] xaddr;  
wire [2:0] yaddr;  
wire [2:0] zaddr;
wire clk, reset;
wired1, n2, n3, n4, n5, n6, n7;
wired[265:0] cbar_data_in;
wired[6:0] sw_req;
wired[48:0] port_req;
wired[6:0] sw_grant;
wired[6:0] data_sent;
wired[48:0] sw_cntrl;

CORNER_PAD corner_0_inst ();
IN_PAD data_in_0_inst   ( .PAD(data_in_pad[0]), .O(data_in[0]) );
IN_PAD data_in_1_inst   ( .PAD(data_in_pad[1]), .O(data_in[1]) );
// ...
// Repeat
// ...
IN_PAD data_in_137_inst   ( .PAD(data_in_pad[137]), .O(data_in[137]) );
IN_PAD data_in_138_inst   ( .PAD(data_in_pad[138]), .O(data_in[138]) );
VDD_PAD vdd_0_inst ();
VSS_PAD vss_0_inst ();
CORNER_PAD corner_1_inst();
IN_PAD data_in_139_inst ( .PAD(data_in_pad[139]), .O(data_in[139]) );
IN_PAD data_in_140_inst ( .PAD(data_in_pad[140]), .O(data_in[140]) );
IN_PAD data_in_141_inst ( .PAD(data_in_pad[141]), .O(data_in[141]) );
// ...
// Repeat
// ...
IN_PAD data_in_264_inst ( .PAD(data_in_pad[264]), .O(data_in[264]) );
IN_PAD data_in_265_inst ( .PAD(data_in_pad[265]), .O(data_in[265]) );
OUT_PAD data_out_0_inst ( .PAD(data_out_pad[0]), .I(data_out[0]) );
OUT_PAD data_out_1_inst ( .PAD(data_out_pad[1]), .I(data_out[1]) );
OUT_PAD data_out_2_inst ( .PAD(data_out_pad[2]), .I(data_out[2]) );
OUT_PAD data_out_3_inst ( .PAD(data_out_pad[3]), .I(data_out[3]) );
OUT_PAD data_out_4_inst ( .PAD(data_out_pad[4]), .I(data_out[4]) );
OUT_PAD data_out_5_inst ( .PAD(data_out_pad[5]), .I(data_out[5]) );
OUT_PAD data_out_6_inst ( .PAD(data_out_pad[6]), .I(data_out[6]) );
OUT_PAD data_out_7_inst ( .PAD(data_out_pad[7]), .I(data_out[7]) );
OUT_PAD data_out_8_inst ( .PAD(data_out_pad[8]), .I(data_out[8]) );
OUT_PAD data_out_9_inst ( .PAD(data_out_pad[9]), .I(data_out[9]) );
OUT_PAD data_out_10_inst ( .PAD(data_out_pad[10]), .I(data_out[10]) );
OUT_PAD data_out_11_inst ( .PAD(data_out_pad[11]), .I(data_out[11]) );
VDD_PAD vdd_1_inst ();
VSS_PAD vss_1_inst ();

CORNER_PAD corner_2_inst ();
OUT_PAD data_out_12_inst (.PAD(data_out_pad[12]), .I(data_out[12]) );
OUT_PAD data_out_13_inst (.PAD(data_out_pad[13]), .I(data_out[13]) );
OUT_PAD data_out_14_inst (.PAD(data_out_pad[14]), .I(data_out[14]) );
// ...
// Repeat
// ...
OUT_PAD data_out_146_inst (.PAD(data_out_pad[146]), .I(data_out[146]) );
OUT_PAD data_out_147_inst (.PAD(data_out_pad[147]), .I(data_out[147]) );
OUT_PAD data_out_148_inst (.PAD(data_out_pad[148]), .I(data_out[148]) );
OUT_PAD data_out_149_inst (.PAD(data_out_pad[149]), .I(data_out[149]) );
OUT_PAD data_out_150_inst (.PAD(data_out_pad[150]), .I(data_out[150]) );
VDD_PAD vdd_2_inst ();
VSS_PAD vss_2_inst ();
Step 1: Modify router_LAXYZ.vnet (5/6)

```
OUT_PAD data_out_264_inst ( .PAD(data_out_pad[264]), .I(data_out[264]) );
OUT_PAD data_out_265_inst ( .PAD(data_out_pad[265]), .I(data_out[265]) );

IN_PAD stop_in_0_inst   ( .PAD(stop_in_pad[0]), .O(stop_in[0]) );
IN_PAD stop_in_1_inst   ( .PAD(stop_in_pad[1]), .O(stop_in[1]) );
IN_PAD stop_in_2_inst   ( .PAD(stop_in_pad[2]), .O(stop_in[2]) );
IN_PAD stop_in_3_inst   ( .PAD(stop_in_pad[3]), .O(stop_in[3]) );
IN_PAD stop_in_4_inst   ( .PAD(stop_in_pad[4]), .O(stop_in[4]) );
IN_PAD stop_in_5_inst   ( .PAD(stop_in_pad[5]), .O(stop_in[5]) );
IN_PAD stop_in_6_inst   ( .PAD(stop_in_pad[6]), .O(stop_in[6]) );

OUT_PAD stop_out_0_inst ( .PAD(stop_out_pad[0]), .I(stop_out[0]) );
OUT_PAD stop_out_1_inst ( .PAD(stop_out_pad[1]), .I(stop_out[1]) );
OUT_PAD stop_out_2_inst ( .PAD(stop_out_pad[2]), .I(stop_out[2]) );
OUT_PAD stop_out_3_inst ( .PAD(stop_out_pad[3]), .I(stop_out[3]) );
OUT_PAD stop_out_4_inst ( .PAD(stop_out_pad[4]), .I(stop_out[4]) );
OUT_PAD stop_out_5_inst ( .PAD(stop_out_pad[5]), .I(stop_out[5]) );
OUT_PAD stop_out_6_inst ( .PAD(stop_out_pad[6]), .I(stop_out[6]) );
```
IN_PAD xaddr_0_inst   ( .PAD(xaddr_pad[0]), .O(xaddr[0]) );
IN_PAD xaddr_1_inst   ( .PAD(xaddr_pad[1]), .O(xaddr[1]) );
IN_PAD xaddr_2_inst   ( .PAD(xaddr_pad[2]), .O(xaddr[2]) );
IN_PAD yaddr_0_inst   ( .PAD(yaddr_pad[0]), .O(yaddr[0]) );
IN_PAD yaddr_1_inst   ( .PAD(yaddr_pad[1]), .O(yaddr[1]) );
IN_PAD yaddr_2_inst   ( .PAD(yaddr_pad[2]), .O(yaddr[2]) );
IN_PAD zaddr_0_inst   ( .PAD(zaddr_pad[0]), .O(zaddr[0]) );
IN_PAD zaddr_1_inst   ( .PAD(zaddr_pad[1]), .O(zaddr[1]) );
IN_PAD zaddr_2_inst   ( .PAD(zaddr_pad[2]), .O(zaddr[2]) );
IN_PAD clk_inst       ( .PAD(clk_pad),      .O(clk ) );
IN_PAD reset_inst     ( .PAD(reset_pad),    .O(reset) );
VDD_PAD vdd_3_inst ();
VSS_PAD vss_3_inst ();

              .A4(n3), .ZN(data_sent[4]) );
              .A4(data_out[158]), .ZN(n3) );
// ...
Step 2:
Make router_LAXYZ.io(1/5)

```plaintext
(globals
    version = 3
    io_order = default
)
iopad
(iopad
    (topleft
        (inst name="corner_0_inst")
    )
    (top
        (inst name="data_in_0_inst")
        (inst name="data_in_1_inst")
        //... repeat ...
        (inst name="data_in_137_inst")
        (inst name="data_in_138_inst")
        (inst name="vdd_0_inst")
        (inst name="vss_0_inst")
    )
    (topright
        (inst name="corner_1_inst")
    )
)
```
Step 2:
Make router_LAXYZ.io(2/5)

| (right
|  (inst name="data_in_139_inst")
|  (inst name="data_in_140_inst")
|  // ... repeat ...
|  (inst name="data_in_264_inst")
|  (inst name="data_in_265_inst")
|  (inst name="data_out_0_inst")
|  (inst name="data_out_1_inst")
|  (inst name="data_out_2_inst")
|  (inst name="data_out_3_inst")
|  (inst name="data_out_4_inst")
|  (inst name="data_out_5_inst")
|  (inst name="data_out_6_inst")
|  (inst name="data_out_7_inst")
|  (inst name="data_out_8_inst")
|  (inst name="data_out_9_inst")
|  (inst name="data_out_10_inst")
|  (inst name="data_out_11_inst")
|  (inst name="vdd_1_inst")
|  (inst name="vss_1_inst")
| )
Step 2: Make router_LAXYZ.io(3/5)

(bottomright
  (inst name="corner_2_inst")
)
(bottom
  (inst name="data_out_12_inst")
  (inst name="data_out_13_inst")
  // ... repeat ...
  (inst name="data_out_149_inst")
  (inst name="data_out_150_inst")
  (inst name="vdd_2_inst")
  (inst name="vss_2_inst")
)
(bottomleft
  (inst name="corner_3_inst")
)
(left
  (inst name="data_out_151_inst")
  (inst name="data_out_152_inst")
  // ... repeat ...
  (inst name="data_out_264_inst")
  (inst name="data_out_265_inst")
)
Step 2:
Make router_LAXYZ.io(4/5)

(inst name="stop_in_0_inst")
(inst name="stop_in_1_inst")
(inst name="stop_in_2_inst")
(inst name="stop_in_3_inst")
(inst name="stop_in_4_inst")
(inst name="stop_in_5_inst")
(inst name="stop_in_6_inst")
(inst name="stop_out_0_inst")
(inst name="stop_out_1_inst")
(inst name="stop_out_2_inst")
(inst name="stop_out_3_inst")
(inst name="stop_out_4_inst")
(inst name="stop_out_5_inst")
(inst name="stop_out_6_inst")
(inst name="xaddr_0_inst")
(inst name="xaddr_1_inst")
(inst name="xaddr_2_inst")
Step 2: Make router_LAXYZ.io(5/5)

```
  (inst name="yaddr_0_inst")
  (inst name="yaddr_1_inst")
  (inst name="yaddr_2_inst")
  (inst name="zaddr_0_inst")
  (inst name="zaddr_1_inst")
  (inst name="zaddr_2_inst")
  (inst name="clk_inst")
  (inst name="reset_inst")
  (inst name="vdd_3_inst")
  (inst name="vss_3_inst")
```
### Step 1: Setup (File -- Import Design)

```tcl
# setUIVar rda_Input ui_netlist ./input_files/router_LAXYZ.vnet
setUIVar rda_Input ui_timingcon_file ./input_files/router_LAXYZ.sdc
setUIVar rda_Input ui_topcell router_LAXYZ
setUIVar rda_Input ui_leffile {~/lib/cells.lef ~/lib/iopad.lef}
setUIVar rda_Input ui_timelib ~/lib/slow.lib
setUIVar rda_Input ui_io_file ./input_files/router_LAXYZ.io
setUIVar rda_Input ui_pwrnet VDD
setUIVar rda_Input ui_gndnet VSS
setUIVar rda_Input ui_cts_cell_list {CLKBUF_X1 CLKBUF_X2 CLKBUF_X3}
commitConfig
```
# Step 2: Floorplan (Floorplan --> Specify Floorplan)
#
floorPlan -s 150 150 15 15 15 15

saveDesign ./checkpoints/floor.enc

# Step 3: Power ring (Power --> Power Planning --> Add Ring)
#
addRing -nets {VSS VDD} -type core_rings \
  -spacing_top 2 -spacing_bottom 2 -spacing_right 2 -spacing_left 2 \
  -width_top 4 -width_bottom 4 -width_right 4 -width_left 4 \
  -around core -jog_distance 0.095 -threshold 0.095 \
  -layer_top metal10 -layer_bottom metal10 -layer_right metal9 \
  -layer_left metal9 \
  -stacked_via_top_layer metal10 -stacked_via_bottom_layer metal1
Step 3: 

Make `iopad_LAXYZ.tcl` (3/9)

```
# STEP 4: Power stripe (Power --> Power Planning --> Add Striple)
#
addStripe -nets {VSS VDD} -layer metal8 -width 4 -spacing 2 \
    -block_ring_top_layer_limit metal9 -block_ring_bottom_layer_limit metal7 \
    -padcore_ring_top_layer_limit metal9 -padcore_ring_bottom_layer_limit metal7 \
    -stacked_via_top_layer metal10 -stacked_via_bottom_layer metal1 \
    -set_to_set_distance 50 -xleft_offset 50 -merge_stripes_value 0.095 \
    -max_same_layer_jog_length 1.6
```
Step 3:
Make iopad_LAXYZ.tcl(4/9)

#
# Step 5: Power route (Route --> Special Router)
#
globalNetConnect VDD -pin VDD -inst * -type pgpin
globalNetConnect VSS -pin VSS -inst * -type pgpin

sroute -nets {VSS VDD} -layerChangeRange {1 10} \ 
-connect { blockPin padPin padRing corePin floatingStripe } \ 
-blockPinTarget { nearestRingStripe nearestTarget } \ 
-padPinPortConnect { allPort oneGeom } \ 
-checkAlignedSecondaryPin 1 -blockPin useLef -allowJogging 1 \ 
-crossoverViaBottomLayer 1 -allowLayerChange 1 -targetViaTopLayer 10 \ 
-crossoverViaTopLayer 10 -targetViaBottomLayer 1

saveDesign ./checkpoints/power.enc
Step 3: Make *iopad_LAXYZ.tcl*(5/9)

```plaintext
#
# Step 6: Placement (Place --> Standard Cell)
#
placeDesign -prePlaceOpt

#
# Step 7: Optimization (preCTS) (Optimize --> Optimize Design)
#
optDesign -preCTS

#
# Step 8: Clock tree synthesis (CTS) (Clock --> Cynthesize Clock Tree)
#
addCTSCellList {CLKBUF_X1 CLKBUF_X2 CLKBUF_X3}
clockDesign -genSpecOnly Clock.ctstch
clockDesign -specFile Clock.ctstch -outDir clock_report -fixedInstBeforeCTS
saveDesign ./checkpoints/cts.enc
```
# Step 3: Make iopad_LAXYZ.tcl(6/9)

# Step 9: Clock tree check (Clock --> Display --> Display Clock Tree)
#

# Step 9: Optimization (postCTS) (Optimize --> Optimize Design)
#
optDesign -postCTS
optDesign -postCTS -hold
# Step 3: Make *iopad_LAXYZ.tcl*(7/9)

```tcl
# # Step 10: Detailed route (Route --> Nano Route --> Route)
#
setNanoRouteMode -quiet -routeWithTimingDriven true
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven true
routeDesign -globalDetail

# # Step 11: Optimization (postRoute) (Optimize --> Optimize Design)
#
optDesign -postRoute
optDesign -postRoute -hold

saveDesign ./checkpoints/route.enc
```
# Step 12: Add fillers (Place --> Physical Cells --> Add Filler)
#
addFiller -prefix FILLER -cell FILLCELL_X1 FILLCELL_X2 FILLCELL_X4 \
   FILLCELL_X8 FILLCELL_X16 FILLCELL_X32
#
# Step 13: Verification (LVS) (Verify --> Verify Connectivity)
#
verifyConnectivity -type all -error 1000 -warning 50
#
# Step 14: Verification (DRC) (Verify --> Verify Geometry)
#
verifyGeometry
Step 3:
Make iopad_LAXYZ.tcl(9/9)

# Step 15: Data out (Timing --> Extract RC, Timing --> Write SDF, 
# File --> Save --> Netlist)

saveNetlist ./output/router_LAXYZ.vnet
isExtractRCModeSignoff
rcOut -spef ./output/router_LAXYZ.spef
delayCal -sdf ./output/router_LAXYZ.sdf -idealclock

saveDesign ./checkpoints/final.enc
Step 4: Script execution

Type `velocity` --init script/iopad_LAXYZ.tcl to start execute the script

```
%velocity script/par_LAXYZ.tcl
```

Type `win` to start SoC Encounter and visualize the final layout

```
%velocity script/par_LAXYZ.tcl
```
Your final Chip layout will appear on the main window. Congratulations!
• If we zoom in, we can see the connection established between the pin, the pad and the signals wires connected to the input-ports.
• This figure shows the pads insertion for VDD, VSS, and the local input-port (data-in 0~11)
ACKNOWLEDGEMENT

• THIS WORK IS SUPPORTED BY VLSI DESIGN AND EDUCATION CENTER (VDEC), THE UNIVERSITY OF TOKYO, JAPAN, IN COLLABORATION WITH SYNOPSYS, Inc. and CADENCE DESIGN SYSTEMS, Inc.

• THIS WORK IS SUPPORTED BY COMPETITIVE RESEARCH FUNDING, UNIVERSITY OF AIZU, JAPAN, Ref. P12-2013.