# Cadence® RTL-to-GDSII Flow

**Course Version 6.0**

**Lab Manual** **Revision 1.0**

cadence®

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

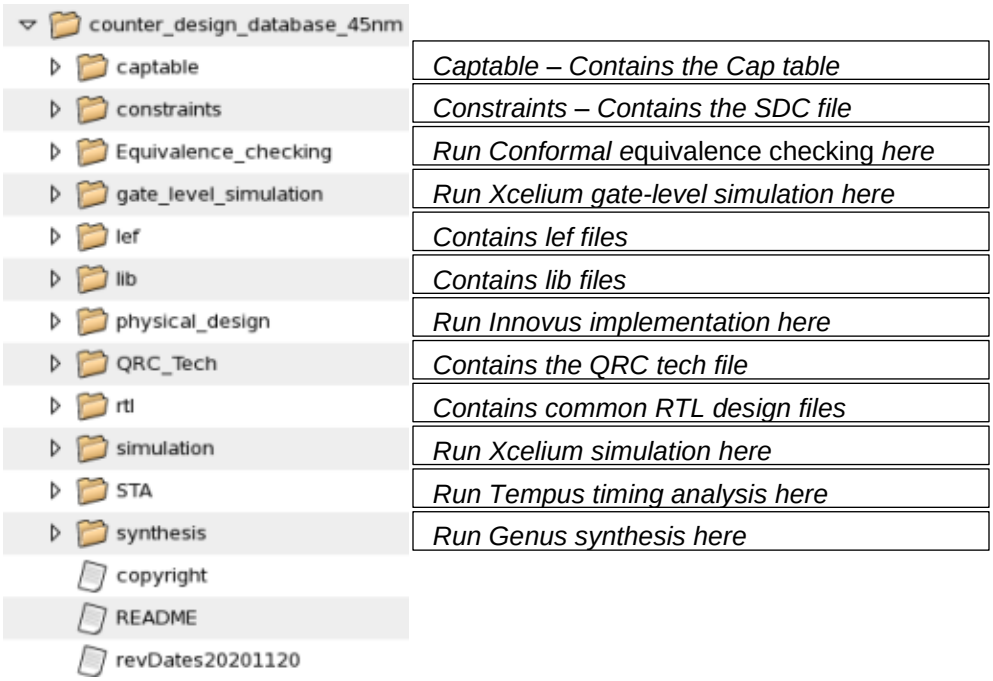
# Table of Contents

## Cadence RTL-to-GDSII Flow

# Module 1: About This Course

### Database Structure

The following is the directory structure of the counter design lab database.




Run the different tools in separate directories so that all log files, command files, and other tool-generated files do not get mixed up.

- DO NOT modify the common directories like **captable**, **constraints**, **lib**, **lef**, **QRC_Tech**, and **rtl**.
- During your run, save your modified design files inside the same directory where you run those tools.

# Module 2: Design Specification and RTL Coding

**There are no labs in this module**

# Module 3: Design Simulation Using the Xcelium Simulator

## Lab 3-1 Simulating a Simple Counter Design

**Objective: To simulate a simple counter design with a testbench using the Xcelium™ Simulator.**

---

This lab uses the following software:

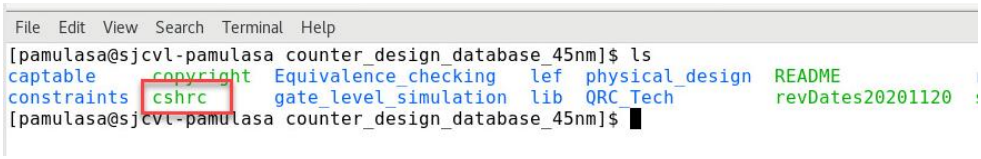
- XCELIUM 23.09 (23.09.001)

### Steps to Invoke Tools

Before invoking any tool, invoke the C shell by entering “csh” in the terminal.

1. Open cshrc file located in the directory *counter_design_database_45nm*, by entering the command.

   ```
   gvim cshrc
   ```




2. Set the tool installation paths by replacing the word ‘*<tool_installation>*’ for each tool in **cshrc** file.

```
#User need to set <tool_installation> to user's tool path


echo "Sourcing Xcelium license"
setenv XLMHOME <tool_installation>/xcelium/2309/23.09.001

echo "Sourcing vManager license to lauch IMC tool"
setenv VMGRHOME <tool_installation>/vmanager/2309/23.09.001

echo "Sourcing Modus License"
setenv MODHOME <tool_installation>/MODUS231/23.10.000

echo "Sourcing Conformal License"
setenv CNFRLHOME <tool_installation>/CONFRML232/23.20.100

echo "Sourcing DDI221 for Genus License"
setenv DDI_GENUS  <tool_installation>/DDI231/23.10-p003_1/GENUS231

echo "Sourcing DDI221 for Innovus License"
setenv DDI_INNOVUS  <tool_installation>/DDI231/23.10-p003_1/INNOVUS231

echo "Sourcing SSVHOME License"
setenv SSVHOME  <tool_installation>/SSV231/23.10-p001_1

 set path = ($XLMHOME/tools/bin \
             $VMGRHOME/bin \
             $MODHOME/lnx86/tools.lnx86/bin   \
             $CNFRLHOME/lnx86/tools.lnx86/bin \
             $DDI_GENUS/tools.lnx86/bin \
             $DDI_INNOVUS/tools.lnx86/bin \
             $SSVHOME/lnx86/tools.lnx86/bin \
             $path )
```

*For more information about the cshrc setting, check the **README** file.*

3. After setting the paths of each tool, source the *cshrc* file by entering the command:

   ```
   source cshrc
   ```

   **Note**: Make sure the cshrc file is sourced without any errors.

### Verification Using Simulation

Cadence® Single-Core Xcelium Simulator is the tool used for verification. Navigate to the simulation directory, where you have kept your RTL and testbench (simulation directory).

We will discuss both the graphical as well as the batch mode of invoking the Xcelium tool for simulation.

Go to the “simulation” directory:

```
cd simulation
```

You will see two files here: *counter.v*, which is the design itself, and *counter_test.v*, which is the testbench, both written in Verilog.

The process of simulation is, in fact, a three-step process:

1. Compilation, which is “checking the syntax and semantics of the code.”

2. Elaboration, which is “creating the design hierarchy and connecting all the signals within the design.”

3. Simulation, which is “simulating the code using the snapshot created during the Elaboration phase.”

   The Xcelium tool gives the xrun command, which performs all three steps seamlessly, and gives out the simulation results.

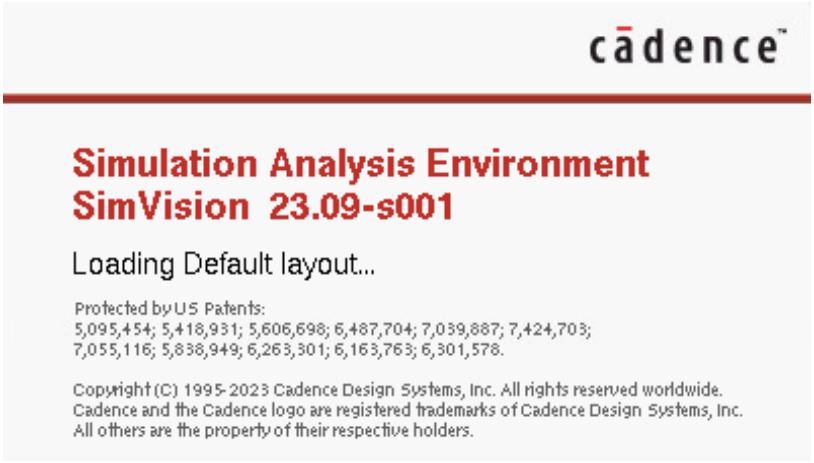
### Using *xrun* in Graphical Mode with the *-gui* Option

1. Execute the following command:

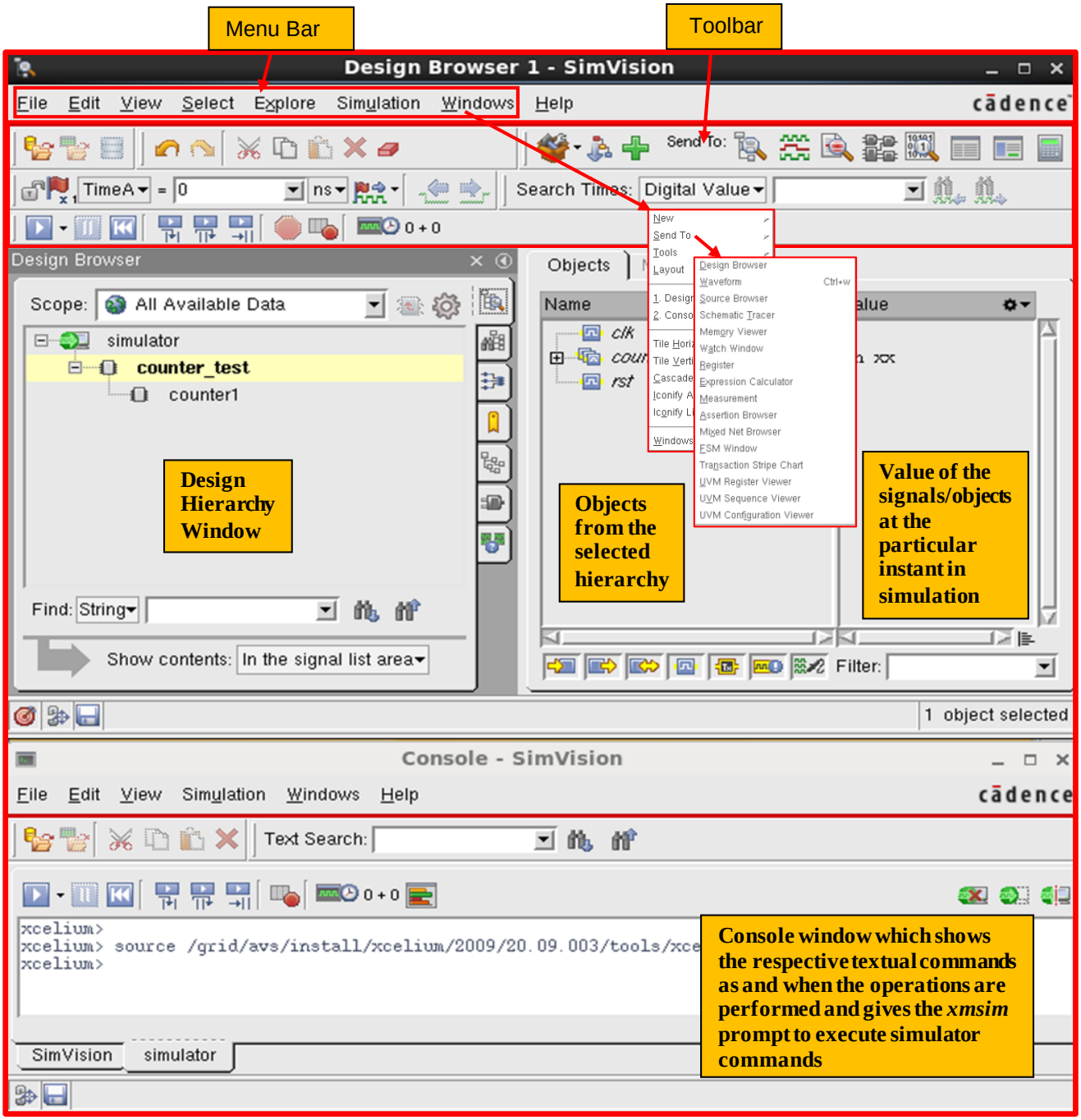
   ```
   xrun counter.v counter_test.v -access +rwc -gui &
   ```

   *-access +rwc* provides probing access to all the signals in the design hierarchy.

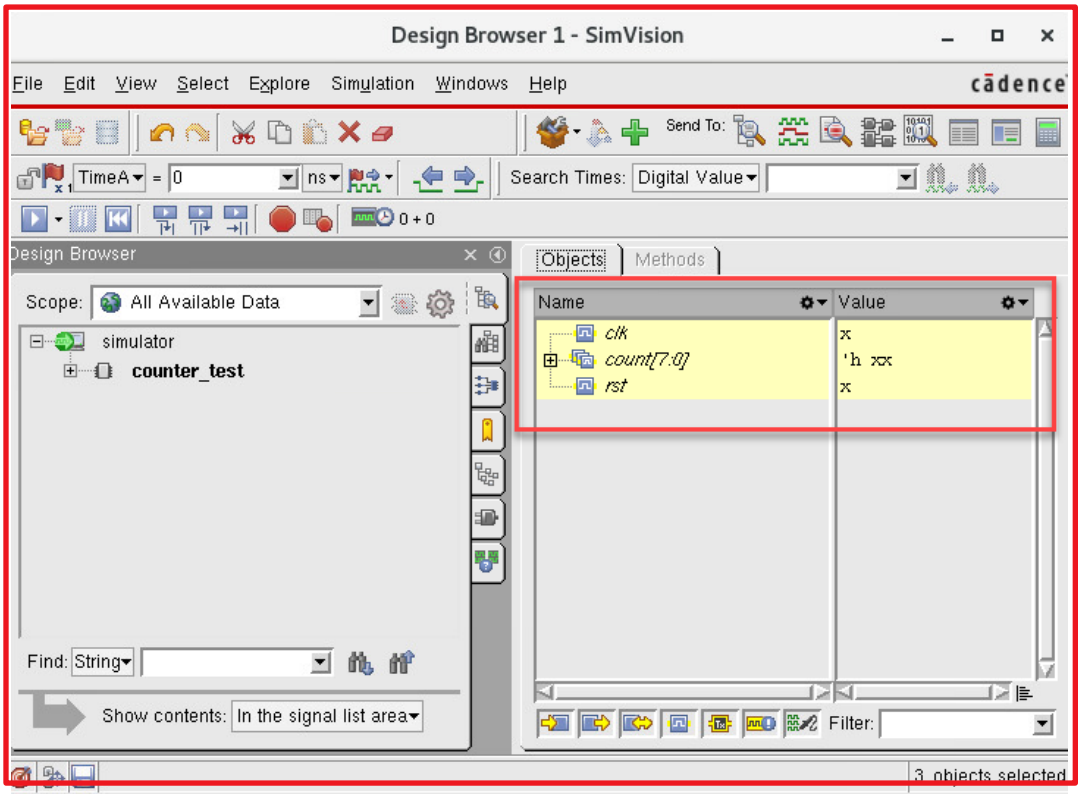
   *-gui* invokes the graphical mode of the Xcelium tool in the following way:



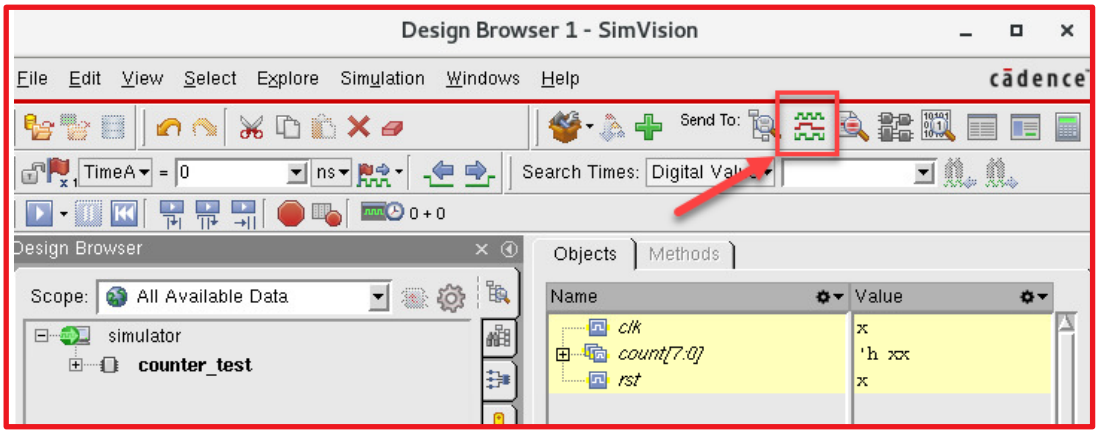
2. It opens the SimVision™ tool with the windows, as shown below.

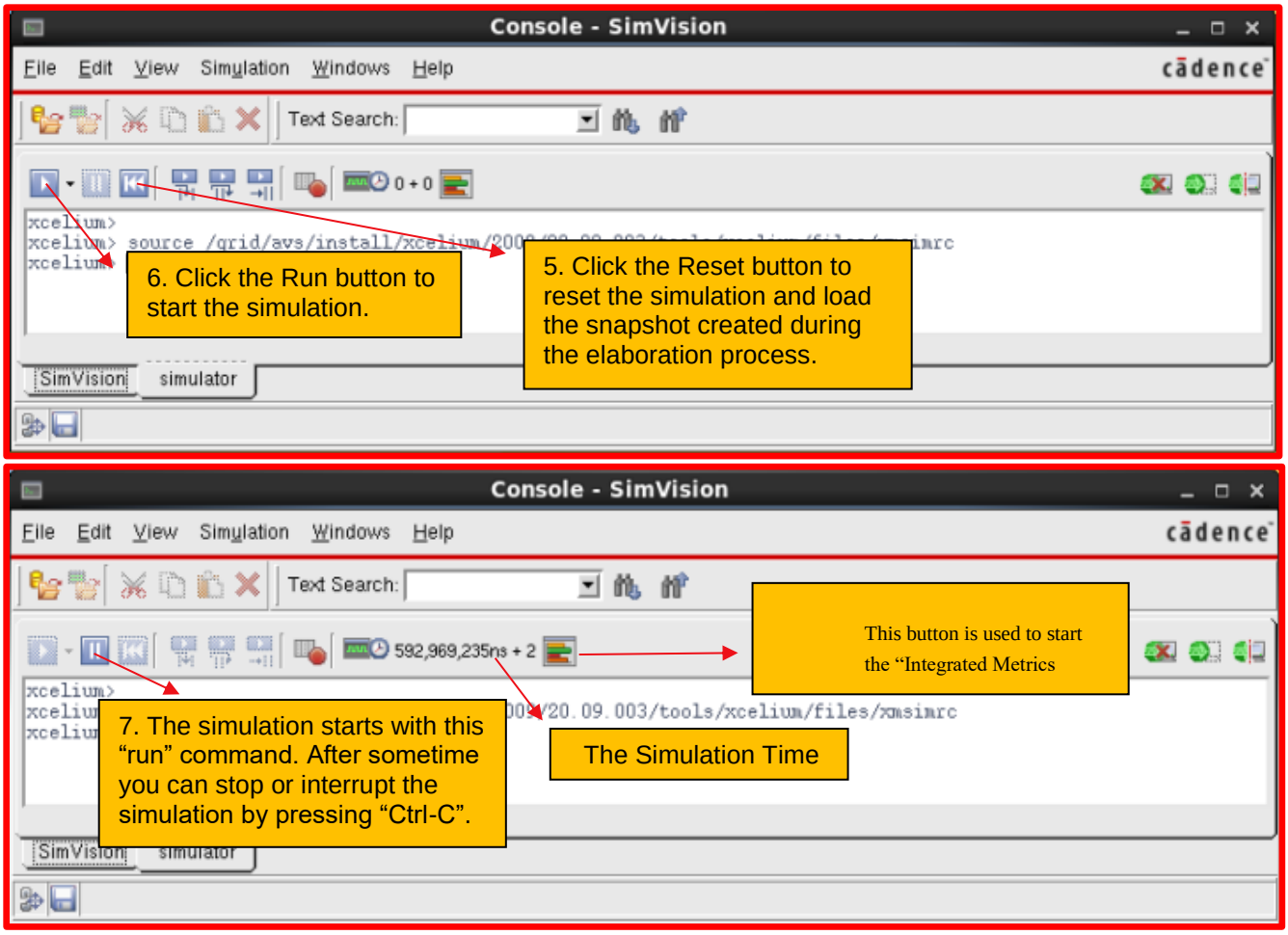


3. Click **counter_test** and select signals from the objects of the *Design Browser* window.

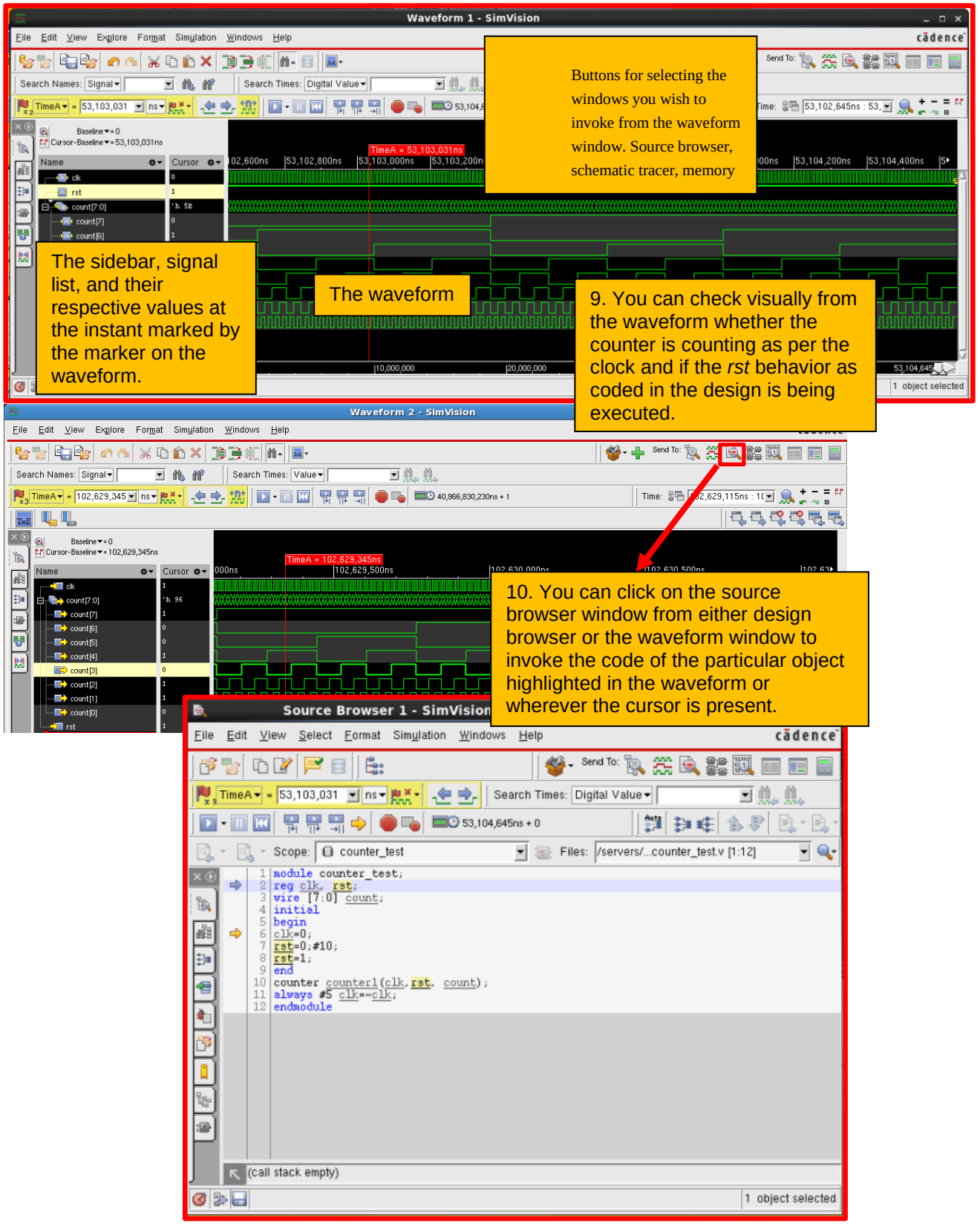



4. After selecting the signals from the design hierarchy, click on the waveform icon on the *Design Browser* window.





5. You can see the waveform below with the simulation running, with selected signals in the window.



## Using *xrun* in Batch Mode

1. Go to the *simulation* directory:

```
cd simulation
```

```
File  Edit  View  Search  Terminal  Tabs  Help
                    pamulasa@sjcvl-pamulasa:simulation
[pamulasa@sjcvl-pamulasa simulation]$ ll
total 32
-rwxr-xr-x 1 pamulasa cadence1  172 Nov 19  2020 cleanup_dirs
-rwxr-xr-x 1 pamulasa cadence1  161 Jun 19  2017 counter_test.v
-rwxr-xr-x 1 pamulasa cadence1  167 Jun 19  2017 counter.v
drwxr-xr-x 2 pamulasa cadence1 4096 Mar 29 22:04 waves.shm
drwxr-xr-x 4 pamulasa cadence1 4096 Mar 29 22:03 xcelium.d
-rw-r--r-- 1 pamulasa cadence1  333 Mar 29 22:03 xrun.history
-rw-r--r-- 1 pamulasa cadence1  162 Mar 30 22:49 xrun.key
-rw-r--r-- 1 pamulasa cadence1 1432 Mar 30 22:49 xrun.log
[pamulasa@sjcvl-pamulasa simulation]$
```

You will see two files here: *counter.v*, which is the design itself written in Verilog, and *counter_test.v*, which is the testbench as before.

2. Execute the following command:

```
xrun counter.v counter_test.v -access +rwc -clean
```

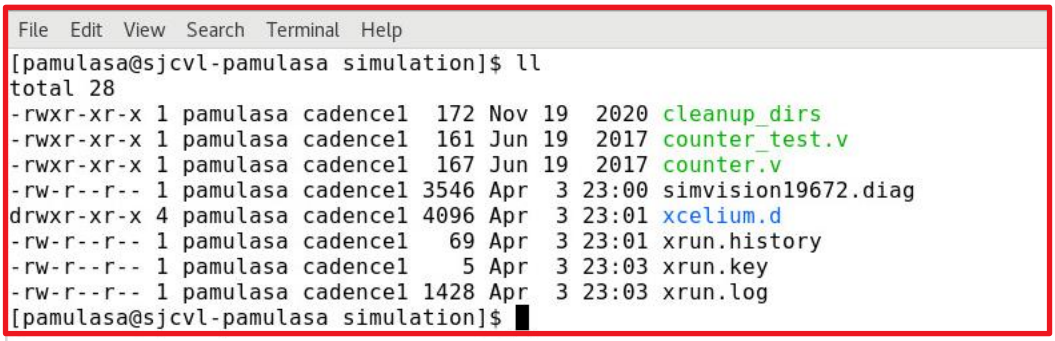
```
[pamulasa@sjcvl-pamulasa simulation]$ ls
cleanup_dirs  counter_test.v  counter.v
[pamulasa@sjcvl-pamulasa simulation]$ xrun counter.v counter_test.v -access +rwc -gui
TOOL:   xrun    23.09-s001: Started on Mar 16, 2024 at 02:44:34 PDT
xrun: 23.09-s001: (c) Copyright 1995-2023 Cadence Design Systems, Inc.
file: counter.v
        module worklib.counter:v
                errors: 0, warnings: 0
file: counter_test.v
        module worklib.counter_test:v
                errors: 0, warnings: 0
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Top level design units:
                counter_test
        Building instance overlay tables: .................... Done
        Generating native compiled code:
                worklib.counter:v <0x7a52e012>
                        streams:   2, words:   417
                worklib.counter_test:v <0x6f029169>
                        streams:   4, words:   996
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                Instances  Unique
                Modules:                2       2
                Registers:              3       3
                Scalar wires:           2       -
                Vectored wires:         1       -
                Always blocks:          2       2
                Initial blocks:         1       1
                Pseudo assignments:     2       -
        Writing initial simulation snapshot: worklib.counter_test:v
xmsim: *W,NOMTDGUI: Multi-Threaded Dumping is disabled for interactive debug mode.

-----------------------------------
Relinquished control to SimVision...
xcelium>
xcelium> source /grid/avs/install/xcelium/2309/23.09.001/tools.lnx86/xcelium/files/xmsimrc
xcelium>
```

Compilation

Elaboration

Simulation

The command creates the file *xcelium.d* as well as the log files in the directory of invocation.




End of Lab

# Module 4: Code Coverage Using the Integrated Metrics Center

# Lab 4-1 Code Coverage Flow for a Simple Counter Design

**Objective: To invoke the Code Coverage tool, the Integrated Metrics Center (or IMC), and analyze the code coverage for a simple counter design with an associated testbench.**

---

In this lab, you use the Integrated Metrics Center software to analyze your code coverage.

This lab uses the following software release:

- XCELIUM 23.09 (23.09.001)
- VERISIUM MANAGER 23.09 (23.09.001)

## What Is Code Coverage?

Coverage is a metric to indicate how well the design is exercised by the testbench. There are two types of coverage:

- Code
- Functional

  Code Coverage assesses how well the tests exercise the design code. It points to areas that did not meet the desired coverage criteria. You can then target those areas with further tests. Code coverage in Incisive® Coverage is classified as the following:

1. Block Coverage – Monitors all exercisable blocks in the Verilog/VHDL source code and identifies unexercised code during simulation.
2. Branch Coverage – Complements block coverage by scoring additional pieces of code that are not, by default, considered individual blocks.
3. Statement Coverage – Provides information on the number of statements within a block.
4. Expression Coverage – Provides information on why a conditional piece of code was executed. It provides statistics for all expressions in the HDL code.
5. Toggle Coverage – Measures the amount of activity in the design, such as unused signals, signals that remain constant, or signals that have too few value changes.
6. FSM Coverage – Interprets the synthesis semantics of the HDL design and monitors the coverage of the FSM representation of control logic blocks in the design.

## Code Coverage Flow

1. Go to the simulation directory of your *counter_design* lab package or database:

   ```
   cd simulation
   ```

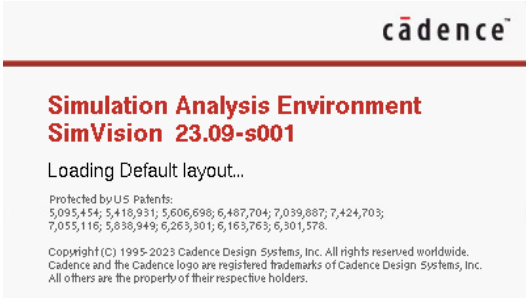
   You will see two files here: *counter.v*, which is the design itself, and *counter_test.v*, which is the testbench, both written in Verilog.

2. Run the following *xrun* command to compile, elaborate, and simulate your counter design, as well as prepare it for coverage analysis by including the *-coverage all* option in the command; this will include all coverage types for analysis.

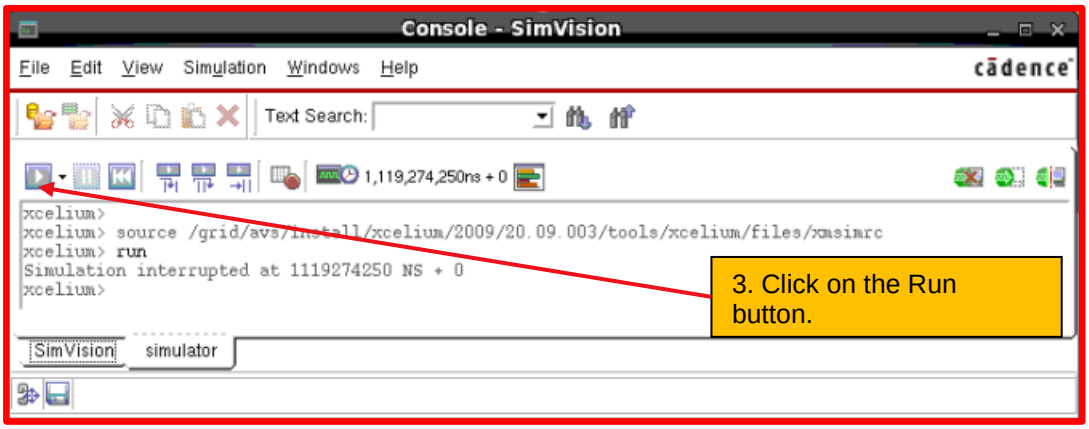
   ```
   xrun counter.v counter_test.v -access +rwc -coverage all -gui
   ```

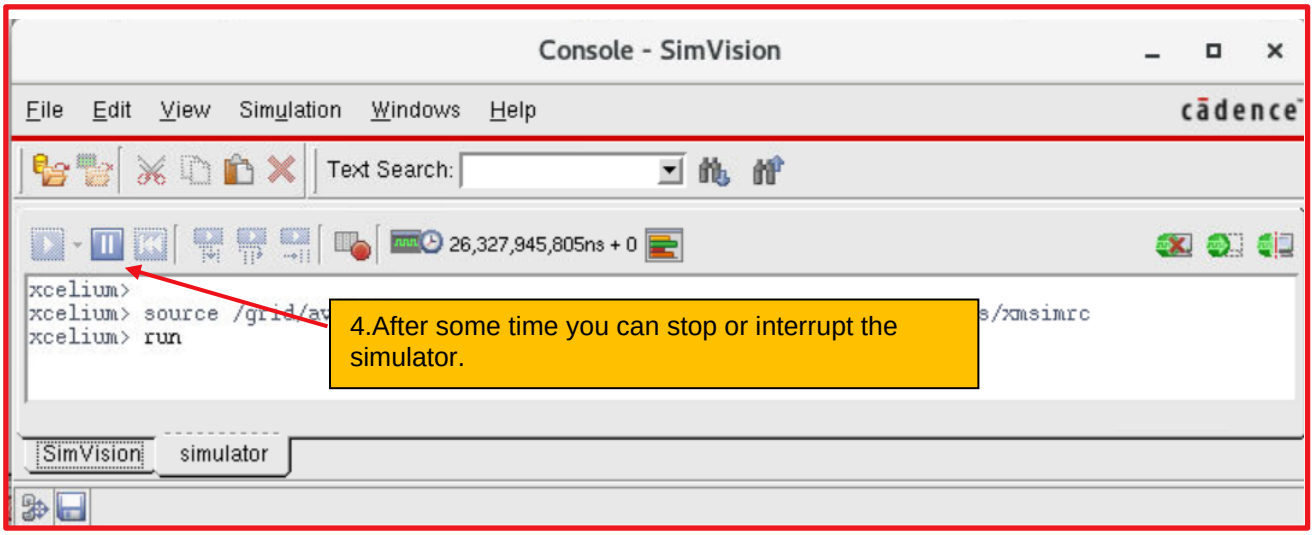
   The command above, along with simulating the design, will create the *cov_work* directory and further create a scope and test directory with coverage model and coverage data, the *.ucm* and *.ucd* files, respectively.

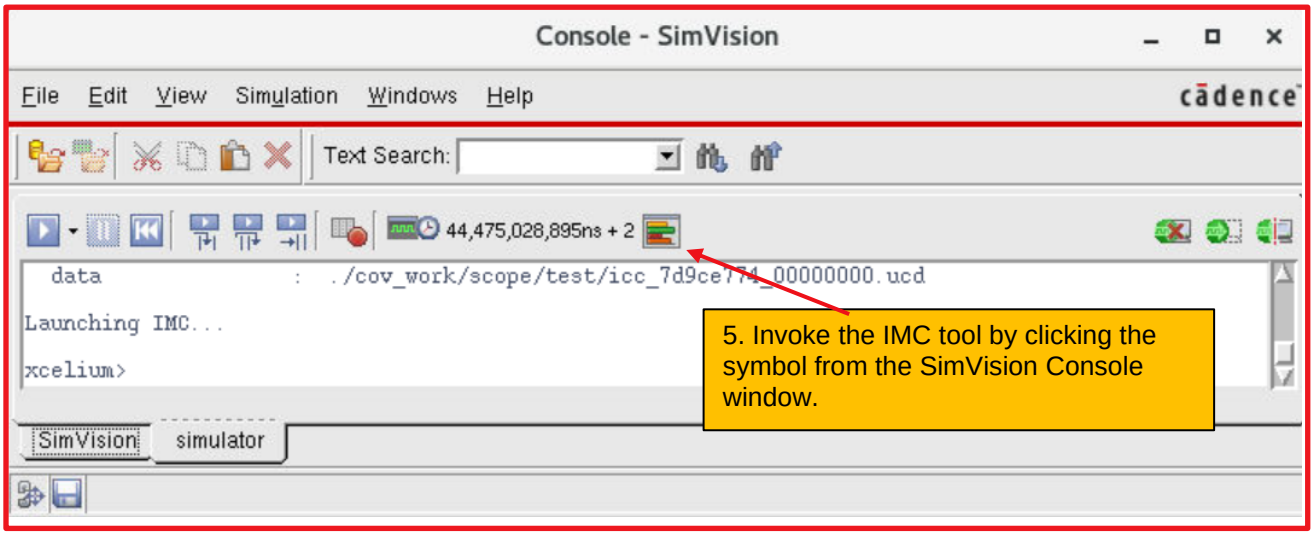
```
[parulh@noivl-parulh simulation]$ ll
total 24
-rwxrwxrwx 1 parulh cadence1  161 Jun 19  2017 counter_test.v
-rwxrwxrwx 1 parulh cadence1  167 Jun 19  2017 counter.v
drwxrwxr-x 3 parulh cadence1 4096 Nov 20 12:06 cov_work
drwxrwxr-x 4 parulh cadence1 4096 Nov 20 12:03 xcelium.d
-rw-rw-r-- 1 parulh cadence1   88 Nov 20 12:03 xrun.history
-rw-rw-r-- 1 parulh cadence1   61 Nov 20 13:34 xrun.key
-rw-rw-r-- 1 parulh cadence1 3413 Nov 20 13:34 xrun.log
```



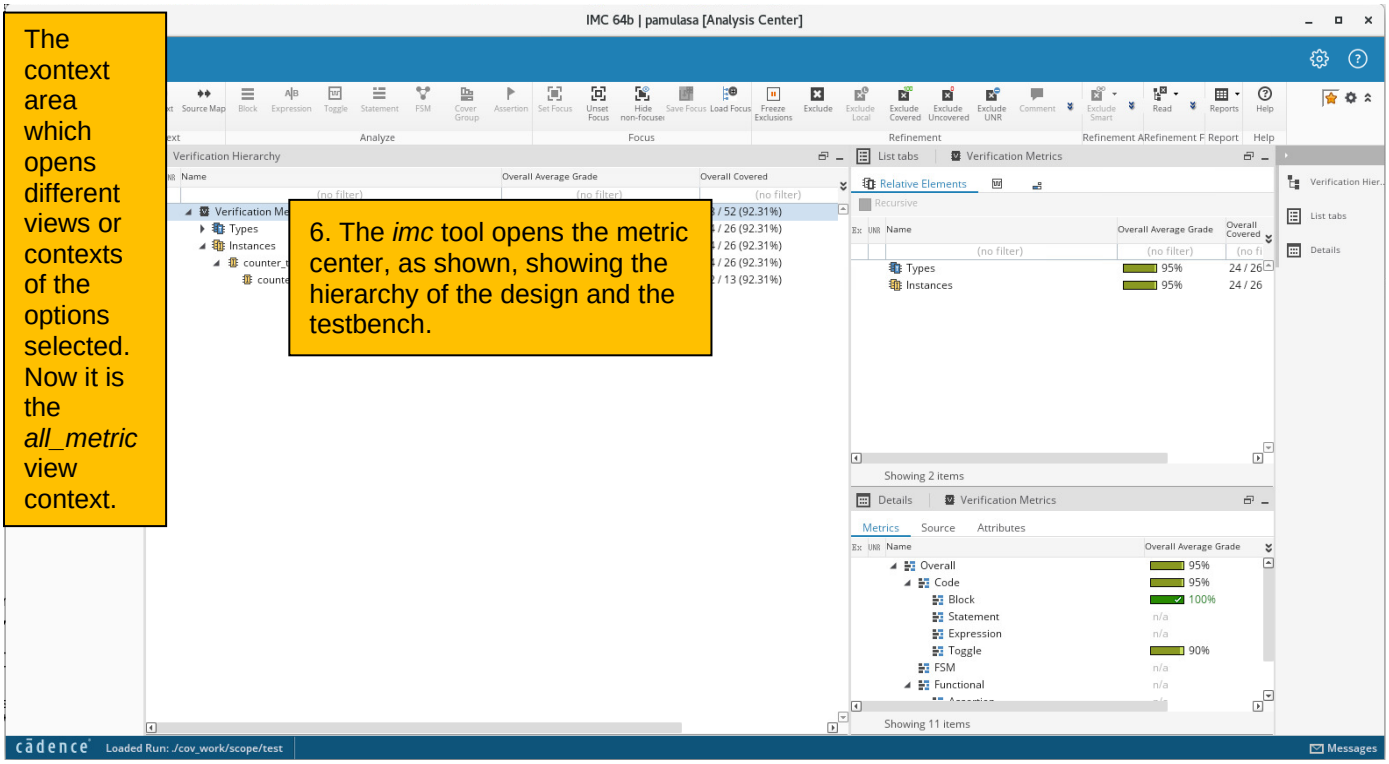

2. The SimVision tool starts because of the *-gui* option in the command and opens up the Console and Design Hierarchy windows.

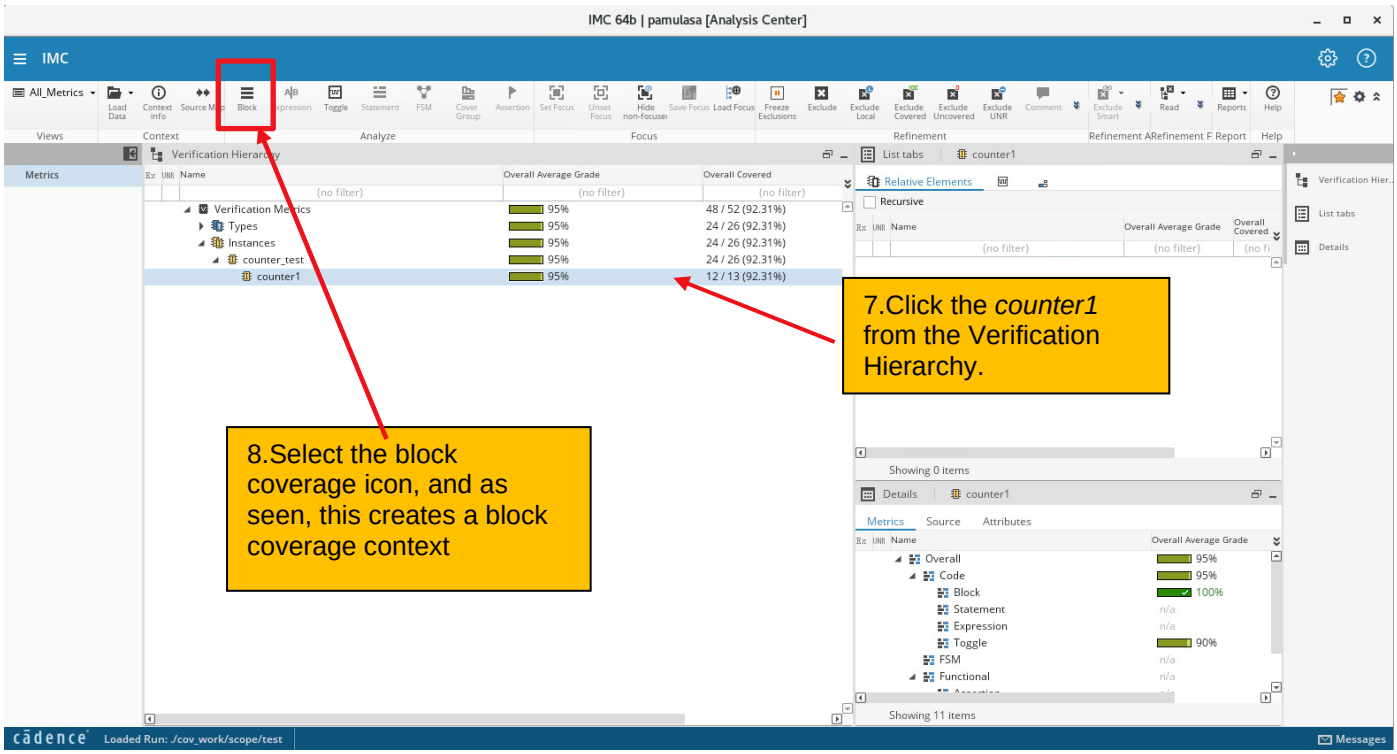


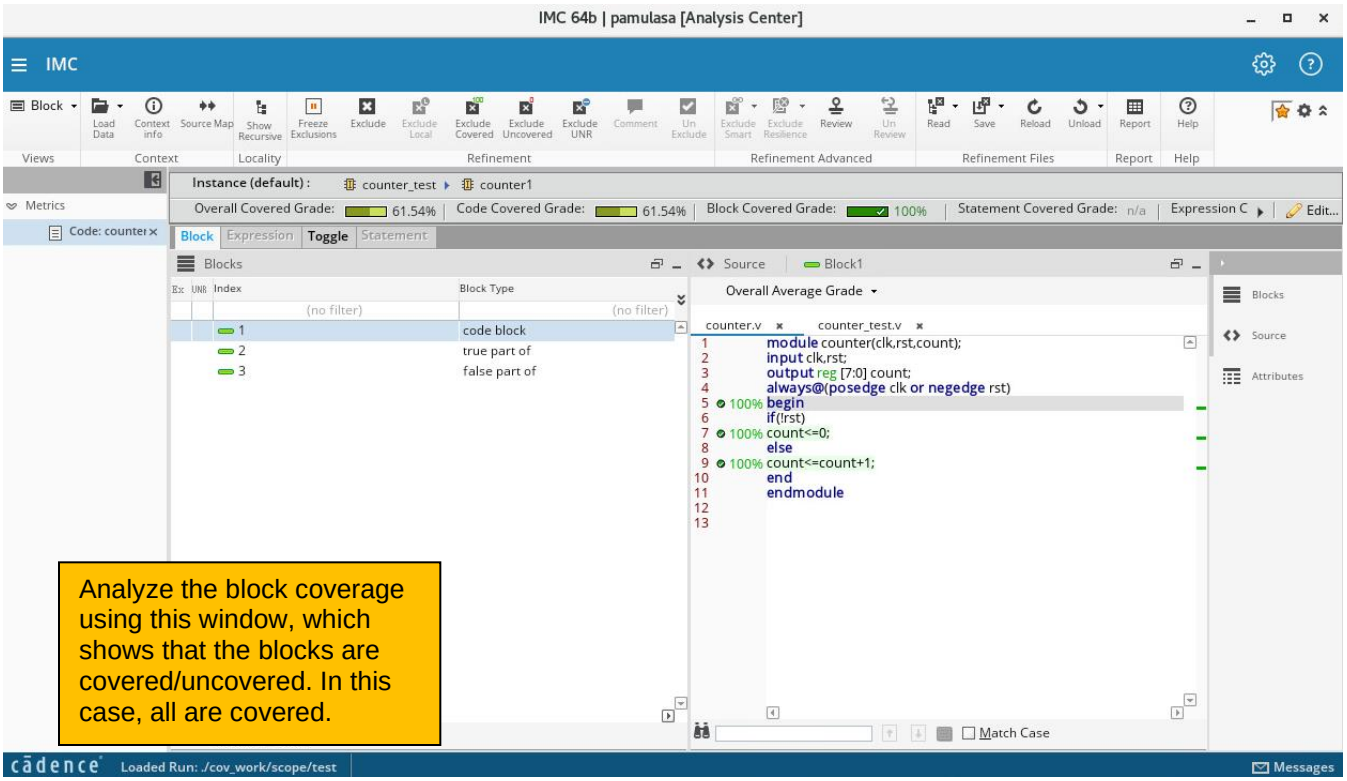


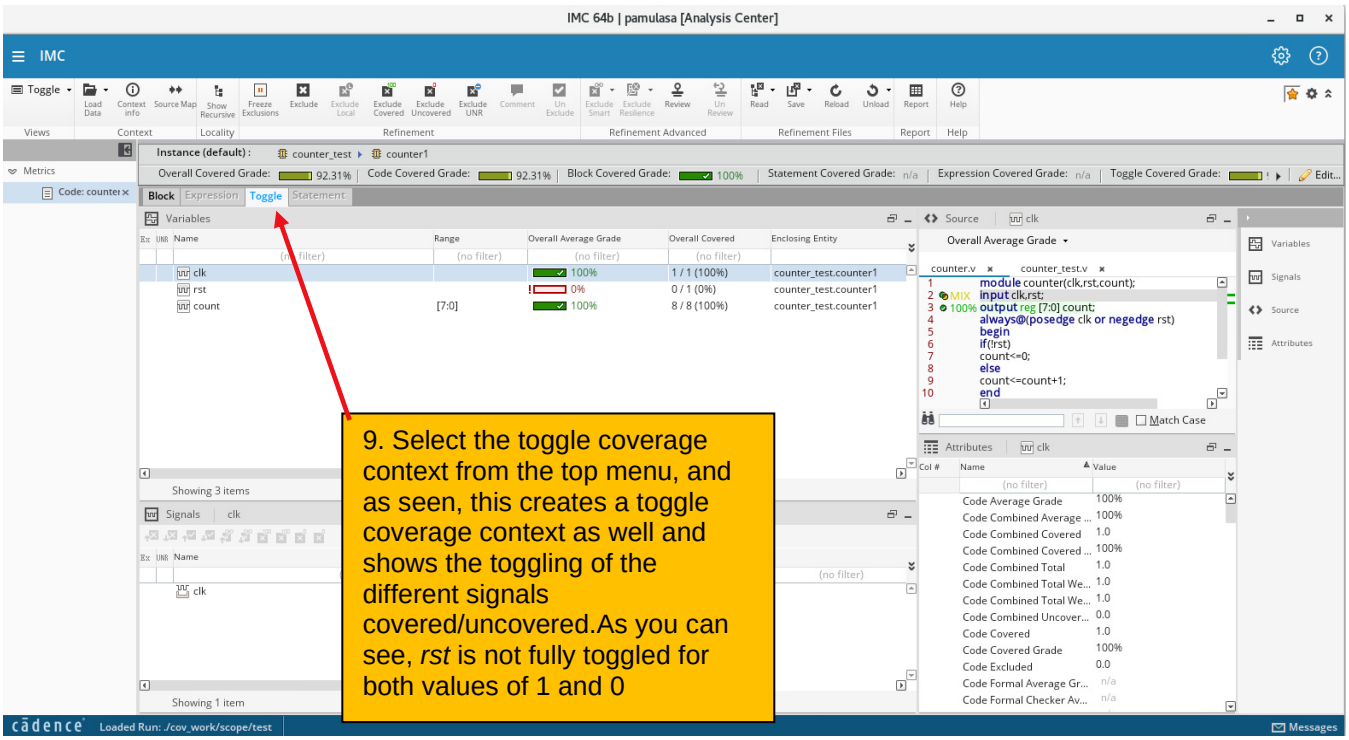




The context area which opens different views or contexts of the options selected. Now it is the *all_metric* view context.






End of Lab

# Module 5: The Synthesis Stage

## Lab 5-1 Running the Basic Synthesis Flow

**Objective: To run the basic synthesis flow for a design.**

---

The tool used for synthesis (converting RTL to a gate-level netlist) is Genus™ Synthesis Solution (Genus) in Stylus Common UI mode.

This lab uses the following software release:

- GENUS231 (23.10-p003_1)

### Steps to Invoke Tools

1. Before invoking any tool, invoke the C shell by entering “csh” in the terminal.

2. Source the *cshrc* file by entering “source <cshrc file>” for example:

```
source cshrc
```

### Running Synthesis (Without DFT)

1. Change the directory to synthesis and write a script file for synthesis.

   Shown below is an example of a script file for synthesis.

```
set_db init_lib_search_path ../lib/
set_db init_hdl_search_path ../rtl/
read_libs slow_vdd1v0_basicCells.lib

read_hdl counter.v
elaborate
read_sdc ../constraints/constraints_top.sdc

set_db syn_generic_effort medium
set_db syn_map_effort medium
set_db syn_opt_effort medium

syn_generic
syn_map
syn_opt

#reports
report_timing > reports/report_timing.rpt
report_power  > reports/report_power.rpt
report_area   > reports/report_area.rpt
report_qor    > reports/report_qor.rpt

#Outputs
write_hdl > outputs/counter_netlist.v
write_sdc > outputs/counter_sdc.sdc
write_sdf -timescale ns -nonegchecks -recrem split -edges check_edge  -setuphold split > outputs/delays.sdf
```

### Creating the Script for Run

The necessary inputs to perform synthesis are RTL, standard cell library, and constraints.

1. To set the search path for libraries, enter:

   ```
   set_db init_lib_search_path <library path>
   ```

   This command will set the path for the standard cell library.

2. To set the search path for HDL/RTL, enter:

   ```
   set_db init_hdl_search_path <rtl path>
   ```

   This command will set the path for *rtl* files.

3. To load the libraries, enter:

   ```
   read_libs <library name>
   ```

   This command will read the specified standard cell library from the specified library path.

4. To read the design, enter:

```
read_hdl <rtl design>
```

   This command will read the *rtl* design.

   **Note:** If the design is hierarchical or has multiple modules instantiated inside the top module, use curly braces “{ }” to mention all modules, including the top design.

   E.g., - read_hdl {top.v sub1.v sub2.v}

   Here *top.v* is the top module, and *sub1.v* and *sub2.v* are the submodules that are instantiated inside the top module.

5. Elaborate the design:

```
elaborate
```

   The *elaborate* command constructs a design hierarchy and connects the signals.

6. To read the constraints, enter:

```
read_sdc  <sdc file name with path>
```

   This command reads in the timing constraints file. Here we have to provide the constraints file name along with the path. An explanation of the constraints file is provided later.

7. To run the synthesis, enter:

```
set_db syn_generic_effort <effort level>
set_db syn_map_effort <effort level>
set_db syn_opt_effort <effort level>
syn_generic
syn_map
syn_opt
```

   These commands will perform synthesis by combining the generic, mapped, and optimization synthesis, and the attributes *syn_generic_effort*, *syn_map_effort,* and *syn_opt_effort* specify the synthesis effort. The effort can be set to “low,” “medium,” or “high,” depending on the scenario.

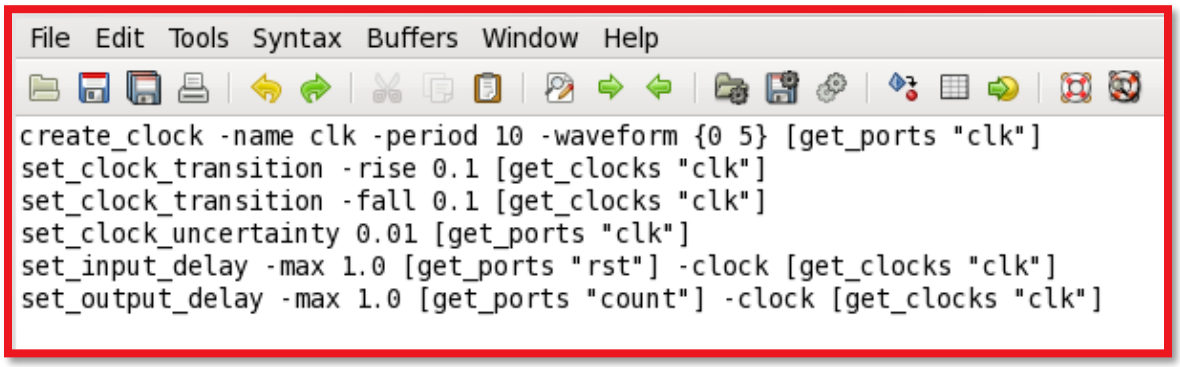
8. Include all the commands mentioned above in the script file.

   **Note:** In counter design, you can see a script file *genus_script.tcl* inside the synthesis directory. Open the script file for further understanding.

### Timing Constraints or the SDC File

Now let us understand the content of the constraints or SDC file.

1. Using SDC, we define clock period, pulse width, rise and fall times, uncertainty and also input and output delays for different signals. The snapshot below contains the constraints file used in the counter design.




2. Let us see the usage and purpose of each command.

   ```
   create_clock –name –period 10 –waveform {0 5} {get_port “clk”}
   ```

   This command will define a clock with a period 10ns, 50% duty cycle, and the signal is high in the first half.

   ```
   set_clock_transition –rise/fall
   ```

   This command defines the transition delay for the clock.

   ```
   set_clock_uncertainty
   ```

   This command will set the uncertainty due to (clock skew and jitter).

   ```
   set_input/output_delay
   ```

   This command will specify the input and output delays used for timing slack calculations.

3. Keep the constraints file inside the constraints directory.

   **Important:** Once the script file to run the synthesis and the constraints file are ready, we can initiate the synthesis. You can either source the complete script or run commands one by one interactively in the Genus Stylus Common UI shell to analyze the synthesis log/results at each stage.

### Starting Genus

1. Change to the *synthesis* directory by entering the following command:

   ```
   cd counter_design_database_45nm/synthesis
   ```

2. Start the software in Stylus Common UI mode by entering:

   ```
   genus
   ```

   **Important:** Use the following command to invoke Genus along with the script file:

   ```
   genus -f  <script file name with path>
   ```

   *genus* is the command to invoke Genus Synthesis Solution, and the *-f* option is used to pass the script to Genus at the time of launching the tool. Genus will execute each command mentioned inside the script file one by one.

   **Note:** If the script file is in the current working directory (synthesis directory), we need not have to provide the path for the script.

   **Note:** In the case of the counter design, the command will be the following:

   ```
   genus –f  genus_script.tcl
   ```

   **Important:** If you are moving ahead by sourcing the complete script in one go, you can skip the next section and continue from the Genus Terminal After Synthesis section.

   **Important:** While performing synthesis, always check the Genus terminal to see whether the tool is reporting any error.

### Loading Libraries and Designs and Synthesizing the Design

1. To set the library and HDL paths, enter the following:

   ```
   set_db init_lib_search_path ../lib/
   set_db init_hdl_search_path ../rtl/
   ```

2. To load the library, enter:

   ```
   read_libs  slow_vdd1v0_basicCells.lib
   ```

3. To read the design, enter:

   ```
   read_hdl counter.v
   ```

4. To elaborate the design, enter:

   ```
   elaborate
   ```

5. To read constraints, enter:

   ```
   read_sdc ../constraints/constraints_top.sdc
   ```

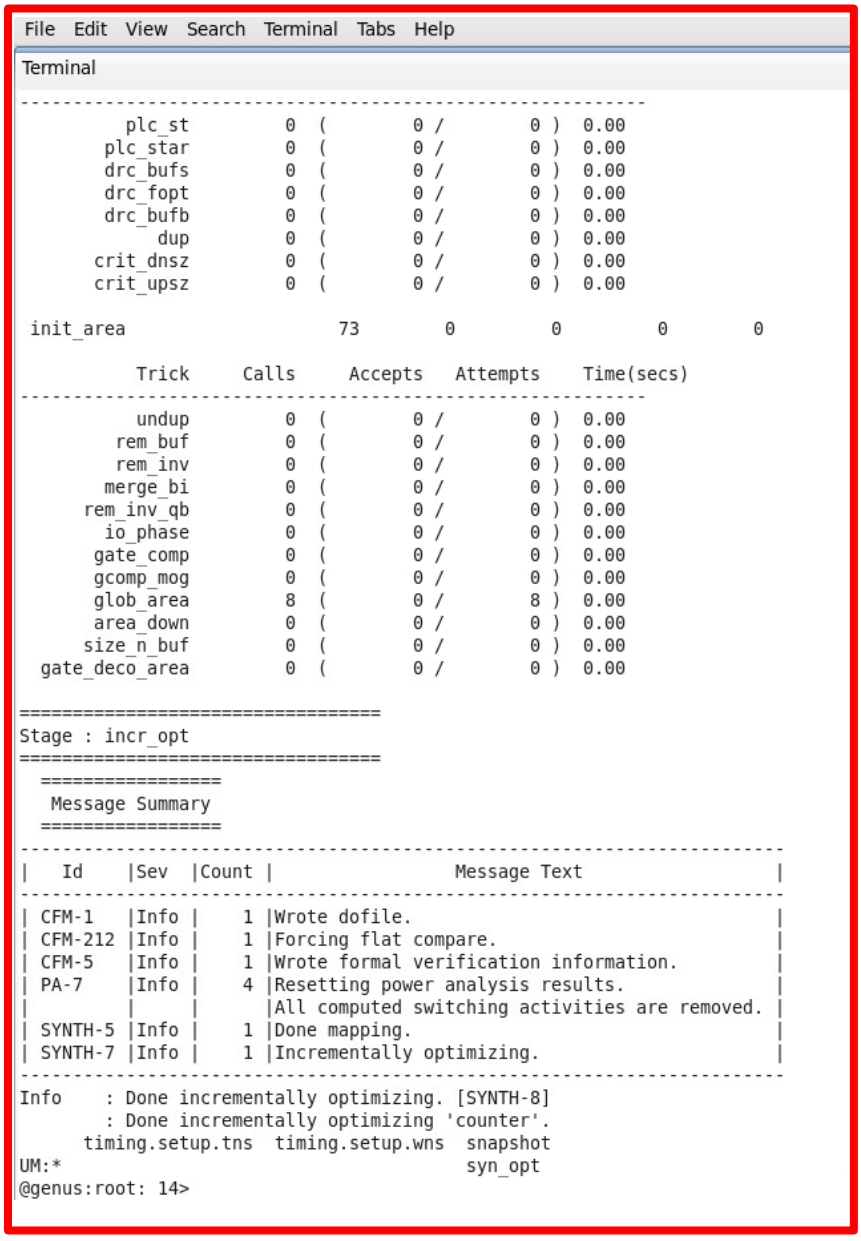
6. Synthesize to generic gates, map to technology library and optimize the design:

   ```
   set_db syn_generic_effort medium
   set_db syn_map_effort medium
   ```

```
set_db syn_opt_effort medium
syn_generic
syn_map
syn_opt
```

## Genus Terminal After Synthesis

The following snapshot shows the Genus Terminal after synthesis.

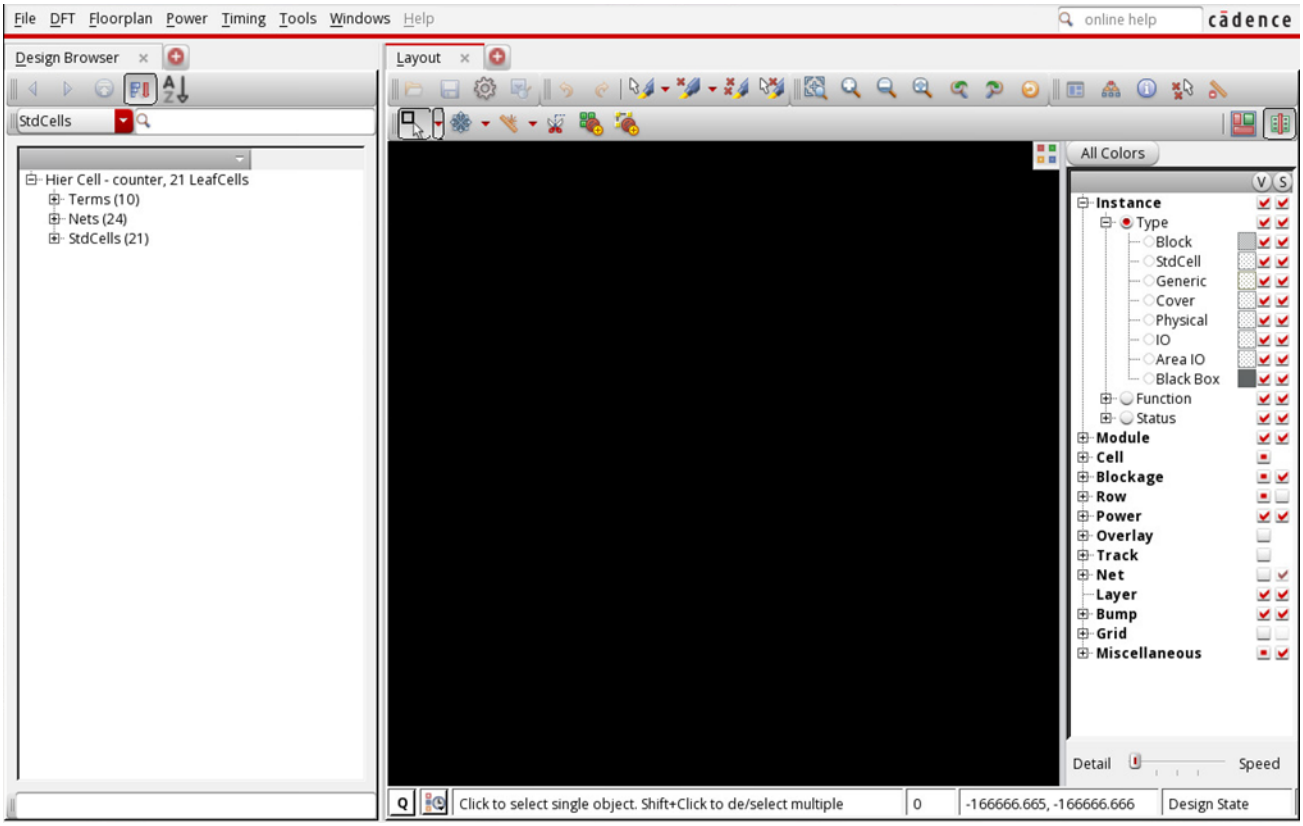


### Launching the GUI

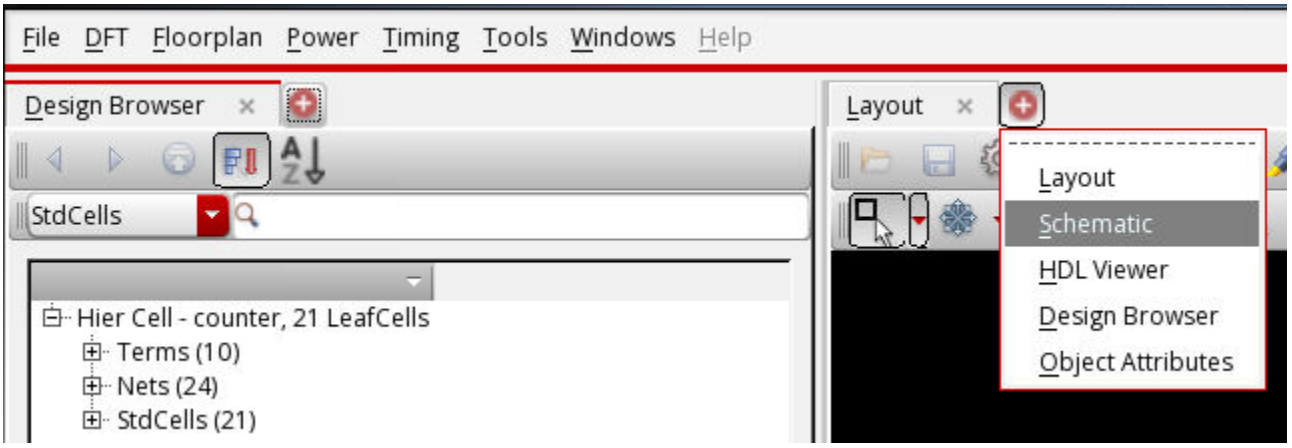
When the synthesis is complete, run the following steps to use the graphical interface.

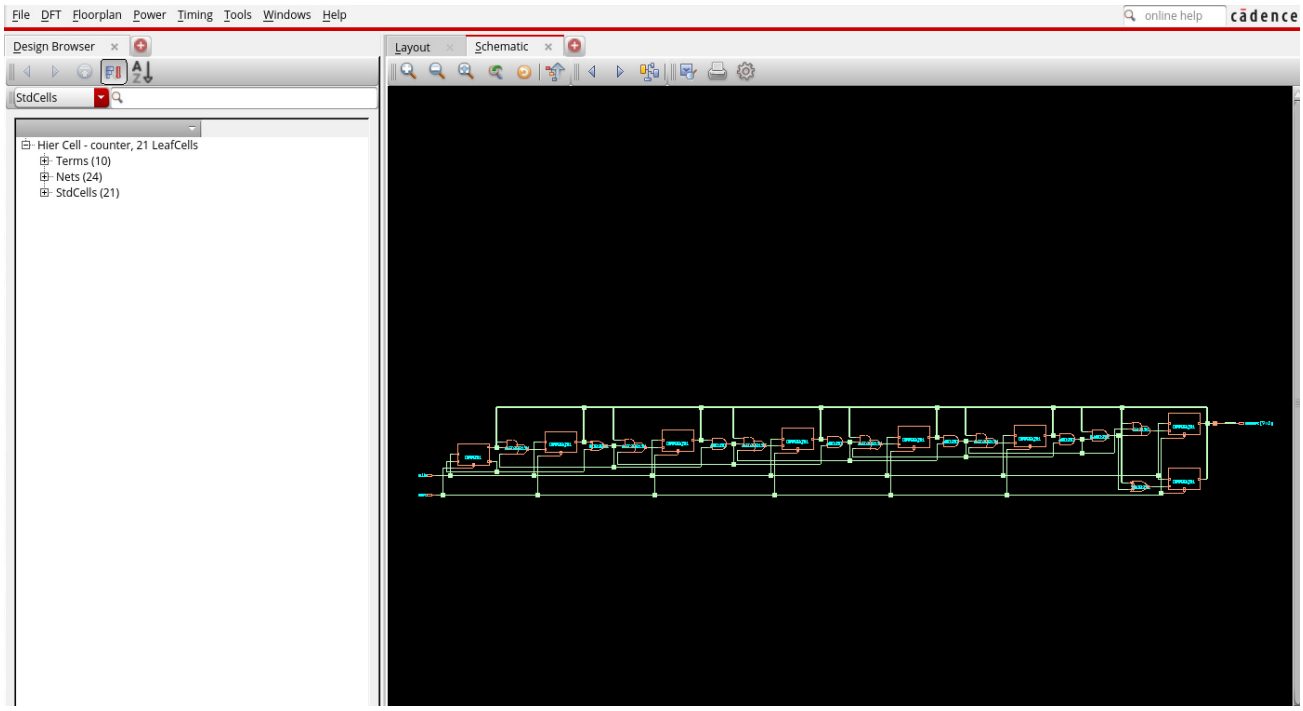
1. View or unhide the graphical interface:

   ```
   gui_show
   ```




2. Click on  to open the Schematic viewer.






3. To close the GUI, enter the following command:

```
gui_hide
```

## Generating Reports

Use the *report_** command to write out the results.

1. To generate a timing report, use:

```
report_timing > reports/report_timing.rpt
```

2. To dump out the power report, use:

```
report_power  > reports/report_power.rpt
```

3. To generate an area report, use:

```
report_area   > reports/report_area.rpt
```

4. To report QOR, use:

```
report_qor    > reports/report_qor.rpt
```

## Writing Output Files

After completing synthesis, use the commands below to dump out a netlist, SDF, SDC, etc., for the next stages of the flow.

1. To write out a synthesized netlist, enter:

```
write_hdl >  outputs/counter_netlist.v
```

2. To generate a final SDC file run, enter:

```
write_sdc >  outputs/counter_sdc.sdc
```

3. To write out an SDF file:

```
write_sdf -timescale ns -nonegchecks -recrem split -edges check_edge  \
   -setuphold split > outputs/delays.sdf
```

   - **timescale**: Mentions the time unit.
   - **nonegchecks**: Ignores the negative timing checks.
   - **recrem**: Splits out the recovery-removal timing check to separate checks for recovery and removal.
   - **edges**: Specifies the edge values.
   - **check_edge**: Keeps edge specifiers on timing check arcs but does not add edge specifiers on combinational arcs.

     **Note:** The commands listed above will generate a netlist, SDF and SDC in the synthesis directory. If you want, you can specify the required directory to save the output files.

## Exiting the Software

1. To close Genus, use the following command:

```
exit
```

   After dumping out the netlist and SDC, we can proceed with Physical Design. Close the synthesis tool.

   **Note:** To know more about synthesis, please refer to the PDFs of the user guide, command reference, attribute reference, etc., available inside the installation directory of the tool.

## Lab 5-2 Running the Synthesis Flow with DFT

**Objective: To run the synthesis flow with DFT for a design.**

In this lab, you will insert scan cells in the design using the Genus Synthesis Solution (Genus) in Stylus Common UI mode.

### Understanding the Flow

Let us look at the files we are working on in this lab.

Change the directory to synthesis and locate **genus_dft_script.tcl**. The main purpose of this script is to set up the variables and other commands that will be used in the flow.

Let us now look at the content of the run script (*genus_dft_script.tcl*). Here is a breakdown of the script flow for clarity:

- Load all the design files and elaborate
- Read SDC (from the constraints directory)
- Read in the DFT setup
- Synthesize to GENERIC
- Synthesize to MAPPED and Optimize

Same as explained in the normal synthesis flow

- Run DFT flow
- Incremental Synthesis
- Write results and database
- Write Modus™ files and ATPG flow

The snapshot below shows *genus_dft_script.tcl*.

```
set_db init_lib_search_path ../lib/
set_db init_hdl_search_path ../rtl/
read_libs slow_vdd1v0_basicCells.lib
read_hdl counter.v
elaborate
read_sdc ../constraints/constraints_top.sdc

set_db dft_scan_style muxed_scan
set_db dft_prefix dft_
define_shift_enable -name SE -active high -create_port SE
check_dft_rules

set_db syn_generic_effort medium
syn_generic
set_db syn_map_effort medium
syn_map
set_db syn_opt_effort medium
syn_opt

check_dft_rules
set_db design:counter .dft_min_number_of_scan_chains 1
define_scan_chain -name top_chain -sdi scan_in -sdo scan_out -create_ports

connect_scan_chains -auto_create_chains
syn_opt -incremental

report_scan_chains
write_dft_atpg -library ../lib/slow_vdd1v0_basiccells.v
write_hdl > outputs/counter_netlist_dft.v
write_sdc > outputs/counter_sdc_dft.sdc
write_sdf -nonegchecks -edges check_edge -timescale ns -recrem split  -setuphold split > outputs/dft_delays.sdf
write_scandef > outputs/counter_scanDEF.scandef
```

## Running Scan Insertion

1. Change to the *synthesis* directory (if you are not in this directory) by entering the following command:

   ```
   cd counter_design_database_45nm/synthesis
   ```

2. Start the software in Stylus Common UI mode by entering:

   ```
   genus
   ```

3. Run the basic setup.

   Set search paths, load the libraries and design, elaborate the design and read constraints:

   ```
   set_db init_lib_search_path ../lib/
   set_db init_hdl_search_path ../rtl/
   read_libs slow_vdd1v0_basicCells.lib
   read_hdl counter.v
   elaborate
   read_sdc ../constraints/constraints_top.sdc
   ```

4. Set the DFT scan flip-flop style for scan replacement using the following command:

   ```
   set_db dft_scan_style muxed_scan
   ```

5. A prefix is added to the name of the DFT logic that is inserted using the following command:

   ```
   set_db dft_prefix dft_
   ```

6. Define the test signals using the following command:

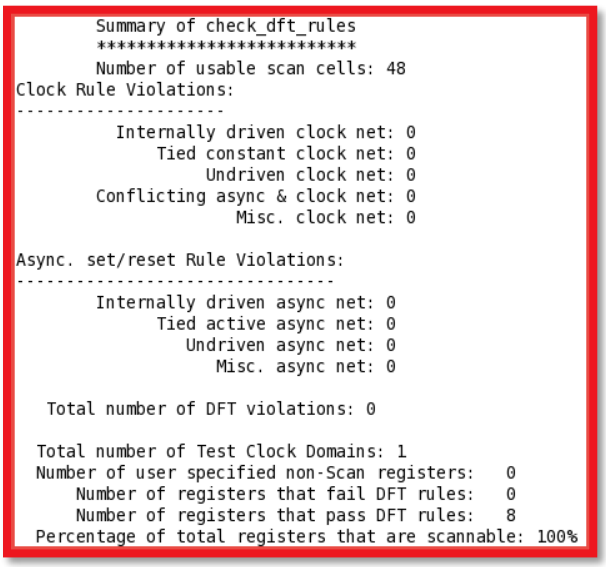
   ```
   define_shift_enable -name SE -active high -create_port SE
   ```

   **Note:** Syntax for the command is given below:

   ```
   define_shift_enable -name {scan_en} -active {high} -create_port
      {scan_en}
   ```

7. It is recommended that you check the DFT rules multiple times during a DFT flow using the following command**:**

```
check_dft_rules
```




As you can see, there are no registers that fail DFT rules, which means that all of the eight registers are eligible for scan connection.

8. Synthesize the design:

```
set_db syn_generic_effort medium
syn_generic
set_db syn_map_effort medium
syn_map
set_db syn_opt_effort medium
syn_opt
check_dft_rules
```

9. Specify the number of scan chains required to connect all FFs using the following command. Here we have used one scan chain:

```
set_db design:counter .dft_min_number_of_scan_chains 1
```

10. Specify the scan-in and scan-out ports of the scan chain using the following command:

```
define_scan_chain -name top_chain -sdi scan_in -sdo scan_out \
-create_ports
```

11. Now connect the scan chains using the *connect_scan_chains* command. This will include all original FFs that were mapped to scan flops.

```
connect_scan_chains -auto_create_chains
```

12. Run incremental synthesis:

```
syn_opt -incremental
```

13. View the DFT chains using the following command:

```
report_scan_chains
```

```
@genus:root: 22> report_scan_chains
Reporting 1 scan chain (muxed_scan)

Chain 1: top_chain
  scan_in:      scan_in
  scan_out:     scan_out
  shift_enable: SE (active high)
  clock_domain: clk (edge: rise)
  length: 8
    bit 1       count_reg[0]  <clk (rise)>
    bit 2       count_reg[1]  <clk (rise)>
    bit 3       count_reg[2]  <clk (rise)>
    bit 4       count_reg[3]  <clk (rise)>
    bit 5       count_reg[4]  <clk (rise)>
    bit 6       count_reg[5]  <clk (rise)>
    bit 7       count_reg[6]  <clk (rise)>
    bit 8       count_reg[7]  <clk (rise)>
-------------------------
```

14. Write out the final netlist, SDF, ScanDEF and constraints using the following commands:

```
write_hdl > outputs/counter_netlist_dft.v
write_sdf -timescale ns -nonegchecks -recrem split -edges check_edge \
   -setuphold split > outputs/dft_delays.sdf
write_sdc > outputs/counter_sdc_dft.sdc
write_scandef > outputs/counter_scanDEF.scandef
```

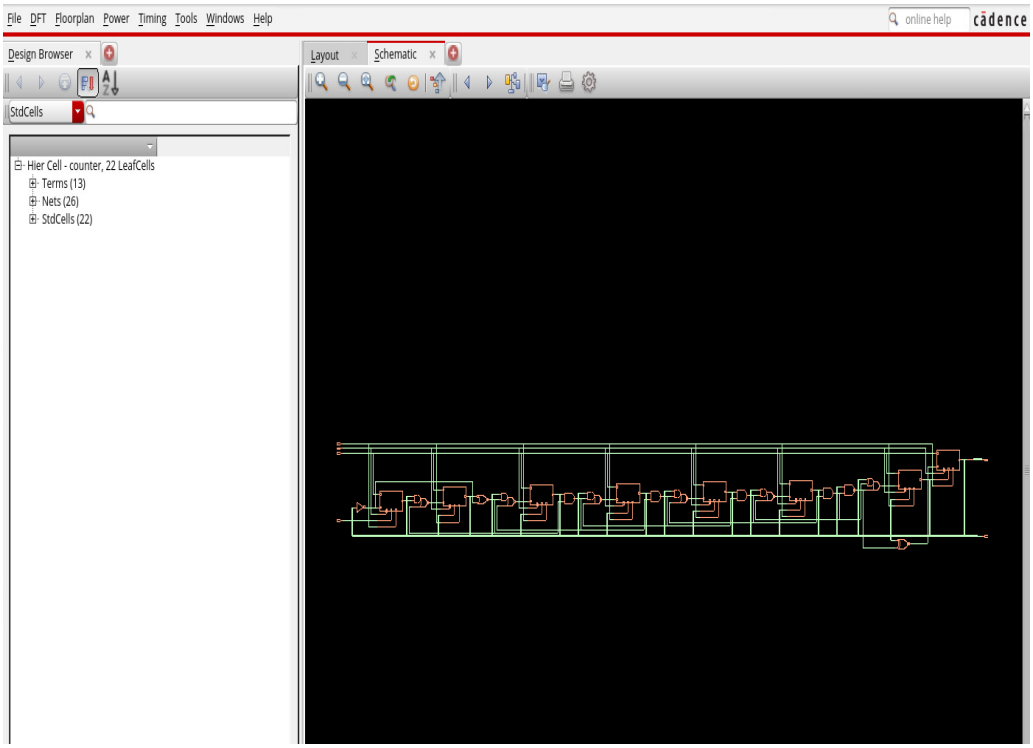
15. We will now run the final ATPG analysis and vector generation. This step will take the final scan chains and run through the basic ATPG flow. This flow is implemented by the following command:

```
write_dft_atpg -library ../lib/slow_vdd1v0_basiccells.v
```

It will generate a directory named *test_scripts* in the current working location.

16. Change the directory to *test_scripts* to see the files that are generated by Genus:
    - counter.test_netlist.v (completed Verilog netlist used for ATPG tool)
    - runmodus.atpg.tcl (ATPG run script)
    - counter.FULLSCAN.pinassign (file specifying I/O test behavior)
    - run_fullscan_sim_sdf (file for running “Back Annotated Simulations” or “Non-zero delay simulations”)
    - run_fullscan_sim (file to run zero delay simulations)

17. View the schematic in the GUI.




18. Exit the software:

```
exit
```

**Note:** You can find solution files (like DFT netlist, ScanDEF, SDC, ATPG scripts etc.) under the *.Solution* folder in the *synthesis* directory.

End of Lab

# Module 6: The Test Stage

## Lab 6-1 Running the Basic ATPG Flow in Modus Test

**Objective: To run the basic ATPG flow for a design in Modus™ Test.**

Modus Test is the tool used to verify the test logic inserted in the netlist during the Synthesis stage.

This lab uses the following software release:

- MODUS231 (23.10.000)

### Steps to Invoke Tools

1. Before invoking any tool, invoke the C shell by entering “csh” in the terminal.

2. Source the *cshrc* file by entering:

   ```
   source <cshrc file>
   ```

   E.g., `source cshrc`

### Creating the Script for Run

1. To build the model, enter:

   ```
   build_model -workdir <directory> -designsource <netlist> \
   -techlib <techlib_files_or_directories> -designtop <top_level_cell>
   ```

   - ***-designsource*** is used to specify the top-level design netlist.
   - ***-designtop*** is used to specify the module definition to model as the Design Under Test (DUT).
   - ***-workdir*** is used to specify the work directory for the run. When Modus Tcl Console is launched, *workdir* automatically gets set to the directory from where the tool is launched. So, specifying -*workdir* is optional for a user.

2. To build test mode, enter:

   ```
   build_testmode -workdir <directory> -testmode <name> \
   -assignfile <filename>
   ```

   - ***-testmode*** is used to specify a name to identify the test mode. *FULLSCAN* is a test mode where the design uses a full scan (no compression) and is suitable for performing static ATPG, timed delay testing, and diagnostics.
   - ***-assignfile*** is used to specify test function pin assignments and other design-specific information.

3. To verify test structures, enter:

   ```
   verify_test_structures -workdir <directory> -testmode <name>
   ```

4. To report test structures, enter:

   ```
   report_test_structures -workdir <directory> -testmode <name>
   ```

5. To build fault model, enter:

   ```
   build_faultmodel -workdir <directory> -fullfault yes|no
   ```

   - ***-fullfault yes*** creates faults for pins on primitives within the technology library cells. The default behavior (if *fullfault* is not specified) is that only the top-level cells will have faults associated with their pins.

6. To create scan test, enter:

   ```
   create_scanchain_tests -workdir <directory> -testmode \
   <testmode> -experiment <exp_name>
   ```

7. To create logic test, enter:

   ```
   create_logic_tests -workdir <directory> -testmode \
   <testmode> -experiment <exp_name> -effort low|high
   ```

   - ***-experiment*** identifies a name to be associated with the output of a test generation or simulation process. The name can be any string that is meaningful to the user. It is used to identify the set of tests in later processing.

8. To write out vectors, enter:

   ```
   write_vectors -workdir <directory> -testmode <testmode> \
      [-inexperiment <exp_nam>] [-language <stil|wgl|verilog|tdl]\
   [-scanformat serial|parallel] - outputfilename <string>
   ```

   - ***-inexperiment*** identifies the data that is to be processed. The name specified is the name that was used for the *-experiment* option in the program that created the data. If the option is not specified, the input data must have been previously processed with *commit_tests (commit_tests* is used to save the logic test experiment created by the *create_logic_tests* command).
   - ***-language*** specifies the converted pattern language format. If not specified, the default is *verilog*.
   - ***-outputfilename*** changes the default output filename.

- ***-scanformat*** is used to specify the format of the scan in the output vectors. Specify *"serial"* to obtain an expanded scan format where values are applied to *scanin* signal pins and measured at *scanout* signal pins. The values are shifted through the scan chains by pulsing the shift clocks. Specify *"parallel"* to obtain a format where scan values are applied directly to and measured directly at the scan registers without actually shifting the values through the scan chains. This format reduces simulation time, but the vectors cannot be applied at a tester. The default is *parallel* when the *language* option is set to *"tdl"*. The default is *"serial"* for all other languages.

### Invoking the Modus Test

1. Change the directory:

   ```
   cd counter_design_database_45nm/synthesis/test_scripts
   ```

2. Invoke the Modus Test tool inside the *test_scripts* directory using the following command:

   ```
   modus
   ```

### Building the Model

1. To build the model use command:

   ```
   build_model -workdir mydir -designsource counter.test_netlist.v \
   -techlib ../../lib/slow_vdd1v0_basiccells.v -designtop counter
   ```

```
@modus:root:/ 1> build_model -workdir mydir -designsource counter.test_netlist.v \
 ==> -techlib ../../lib/slow_vdd1v0_basiccells.v -designtop counter
Starting Command: build_model

INFO (TDA-015): Log File: mydir/testresults/logs/log_build_model_050720171940-026852000 [end TDA_015]

INFO (TEI-195): Build Model - Controller starting: [end TEI_195]


Search Order  1:  "counter.test_netlist.v"
Search Order  2:  "../../lib/slow_vdd1v0_basiccells.v"



Reading Verilog data from counter.test_netlist.v

Verilog Parser complete - 1 modules, 0 gates, 0 user defined primitives.


Reading Verilog data from ../../lib/slow_vdd1v0_basiccells.v

Verilog Parser complete - 546 modules, 1878 gates, 2 user defined primitives.



INFO (TEI-196): Build Model - Hierarchical Model Build starting: [end TEI_196]



Number of blocks in the hierarchical model is: 247.

Number of technology library cell instances in the hierarchical model is: 23.


INFO (TEI-197): Build Model - Hierarchical Model Build completed. [end TEI_197]



INFO (TEI-198): Build Model - Flat Model Build starting: [end TEI_198]

 defaultTIE = X
 defaultDFN = T


INFO (TLM-055): Design Summary
                --------------

Hierarchical Model:                 Flattened Model:
```

```
INFO (TEI-199): Build Model - Flat Model Build completed.  [end TEI_199]


INFO (TDA-001): Maximum Memory used during the run and Cumulative Time in hours:minutes:seconds:

                  Total Memory =           96,759,600  bytes

                      CPU Time =    0:00:01.39
                  Elapsed Time =    0:00:08.80                    [end TDA_001]


INFO (TEI-200): Build Model - Controller completed. [end TEI_200]



--------------------------------------------------------------------------------
*                      Message Summary                                         *
--------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (TEI-195): Build Model - Controller starting:
      1 INFO (TEI-196): Build Model - Hierarchical Model Build starting:
      1 INFO (TEI-197): Build Model - Hierarchical Model Build completed.
      1 INFO (TEI-198): Build Model - Flat Model Build starting:
      1 INFO (TEI-199): Build Model - Flat Model Build completed.
      1 INFO (TEI-200): Build Model - Controller completed.
      1 INFO (TLM-055): Design Summary

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp TDA-009

--------------------------------------------------------------------------------
```

## Building the Test Mode

1. To build the test mode, use command:

```
build_testmode -workdir mydir -testmode FULLSCAN -assignfile \
   counter.FULLSCAN.pinassign
```

```
@modus:root:/ 2> build_testmode -workdir mydir -testmode FULLSCAN -assignfile counter.FULLSCAN.pinassign
Starting Command: build_testmode

INFO (TDA-015): Log File: mydir/testresults/logs/log_build_testmode_FULLSCAN_050720172433-500826000 [end TDA_015]

MODEDEFPATH set to:  /servers/MODUS/lnx86/tools.lnx86/tb/defaults/rules/modedef
TDRPATH set to: /servers/MODUS/lnx86/tools.lnx86/tb/defaults/rules/tdr

Modus Build Test Mode(s) beginning...

   TDR NAME      =  dummy.tdr

   SCAN_TYPE     =  GSD
   BOUNDARY      =  NONE
   IN            =  PI
   OUT           =  PO

INFO (TTM-391): A default modeinit sequence will be generated.   [end TTM_391]
INFO (TTM-387): A default scanop sequence will be generated.   [end TTM_387]
INFO (THM-814): Testmode contains 96.23% active logic,  3.77% inactive logic and  0.00% constraint logic.   [end THM_814]
INFO (TTM-357): There are 1 scan chains which are controllable and observable.   [end TTM_357]

There are no PPIs for Test Mode:  FULLSCAN

Information for Test Mode:  FULLSCAN
-------------------------

         Scan Type = GSD


 Latch Summary for Test Mode:  FULLSCAN
 Latch Type                #Active   #Inactive       Total
 ---------------------   ---------   ---------   ---------
 Test Constraint (TC)            0           0           0
 Scan Enable     (SE)            0           0           0
 Fixed Value     (FV)            0           0           0
 Floating                        0           0           0
 Finite State    (FSM)           0           0           0

INFO (TTM-800): build_testmode has created mode FULLSCAN. [end TTM_800]

Test Function Pin Information for Test Mode:  FULLSCAN
------------------------------------------
```

```
------------------------------------------------------------------------------
*                      Message Summary                                       *
------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (THM-814): Testmode contains 96.23% active logic,  3.77% inactive logic and  0.00% constraint logic.
      1 INFO (TTM-357): There are 1 scan chains which are controllable and observable.
      1 INFO (TTM-387): A default scanop sequence will be generated.
      1 INFO (TTM-391): A default modeinit sequence will be generated.
      1 INFO (TTM-800): build_testmode has created mode FULLSCAN.

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp TDA-009

------------------------------------------------------------------------------
```

## Verifying the Test Structures

It is recommended to verify the test structures before building the fault models so that you can see which fault model is exactly needed to be built.

1. To verify the test structures, use command:

   `verify_test_structures -workdir mydir -testmode FULLSCAN`

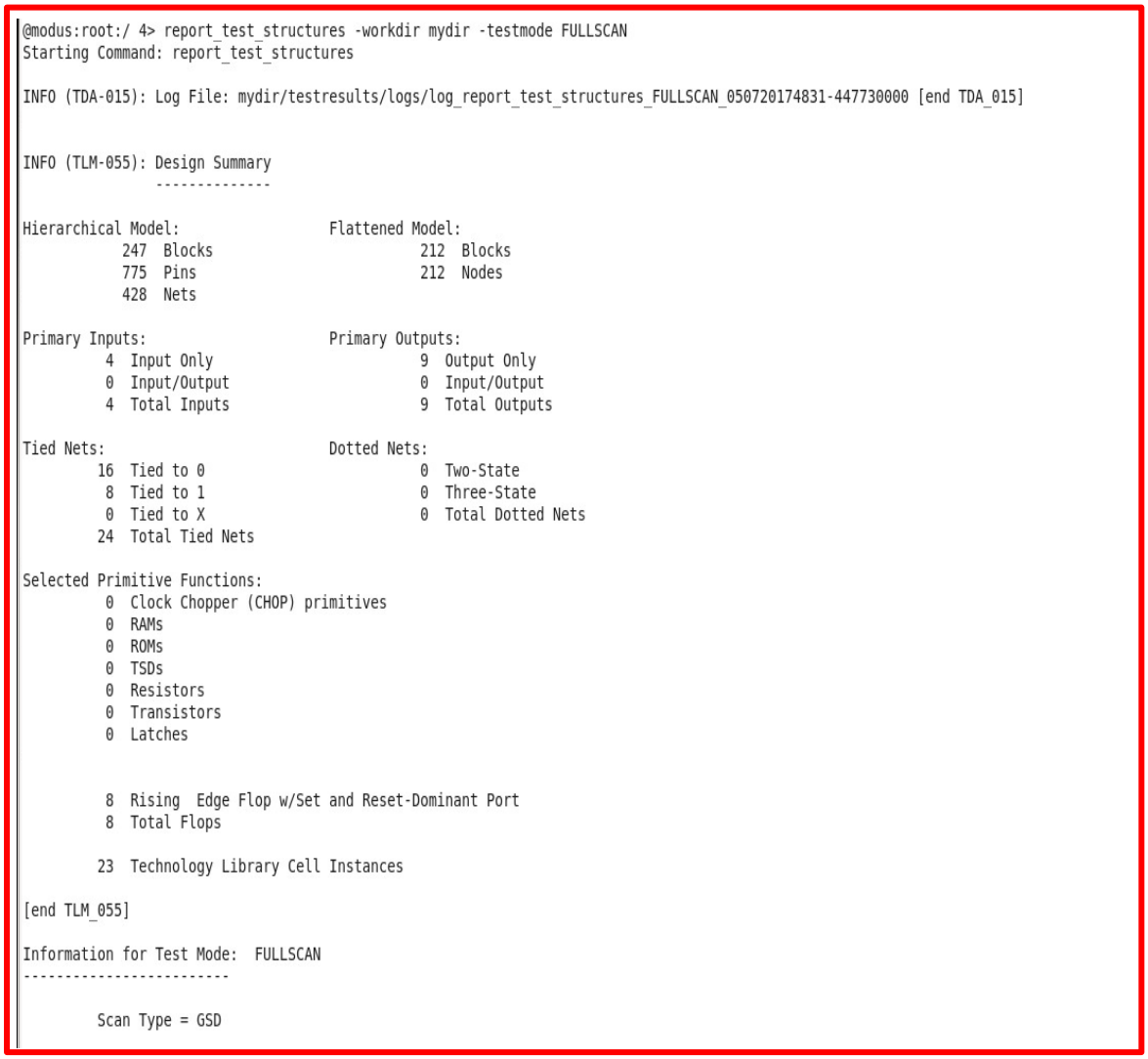
```
@modus:root:/ 3> verify_test_structures -workdir mydir -testmode FULLSCAN
Starting Command: verify_test_structures

INFO (TDA-015): Log File: mydir/testresults/logs/log_verify_test_structures_FULLSCAN_050720173151-388527000 [end TDA_015]

INFO (TSV-900): verify_test_structures processing has started Thu May  7 17:31:51 2020    [end TSV_900]
------------------- verify_test_structures Process Preview ---------------------
  apply constraints ..................................... constraints=yes
  Effort level .......................................... effort=low
  Print 1 message per source of clock race .............. limitcisource=yes
  Print 1 Message per capture of clock race ............. limitcicapture=yes
  Check for mutually exclusive gating logic (MEG) ....... megraces=yes
  Use raise message severity attributes if they exist .... raisemsgsev=no
  Reporting options ..................................... report[option]=yes/no
    Process Preview ....................................   preview=yes
    Circuit Statistics .................................   circuit=yes
    Message Summary ....................................   summary=yes
    Specific Messages ..................................   specific=yes
    Tests Status .......................................   status=no
    Output File Names and Size .........................   filestats=yes
    Display messages for cloaked objects ...............   cloakedobj=no
  Rerun verify_test_structures test(s) ................... reruntests=no
  Use message suppression attributes if they exist ....... suppressmsg=no
  Test selected ......................................... test[option]=yes/no
    Analyze test clock control of memory elements ........   testclockusage=yes
    Analyze three-state drivers for contention ...........   tsdcontention=yes
    Analyze feedback loops and keeper devices ............   feedback=yes
    Analyze clock choppers ...............................   clockchopper=yes
    Analyze flip-flop and latch scan characteristics .....   latchusage=yes
    Analyze explicit fixed value latches .................   explicitfvlatch=yes
    Analyze potential clock signal races .................   clocksignalraces=yes
    Analyze internal scan chains .........................   internalscanchains=yes

--------------------Circuit Statistics--------------------

INFO (TLM-055): Design Summary
                --------------

Hierarchical Model:                  Flattened Model:
            247  Blocks                         212  Blocks
            775  Pins                           212  Nodes
            428  Nets
```

```
*                    Message Summary                                          *
-------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (TLM-055): Design Summary
      1 INFO (TSV-068): The length of the longest scan chain is 8 bit positions, which is 100% of the average scan chain length 8 (based on 8 total scan chain bits and 1 val
id scan chains).
      1 INFO (TSV-378): Scan chain beginning at 'pin scan_in' and ending at 'pin scan_out' is controllable and observable. The length of the scan chain is 8 bit positions.

      1 INFO (TSV-567): There are 1 controllable scan chains fed by Scan In (SI) primary inputs.
      1 INFO (TSV-568): There are 1 observable scan chains feeding to Scan Out (SO) primary outputs.
      1 INFO (TSV-569): There are 0 controllable scan chains fed by on-product Pattern Generator(s).
      1 INFO (TSV-570): There are 0 observable scan chains feeding to on-product Multiple-Input Signature Register (MISRs).
      1 INFO (TSV-900): verify_test_structures processing has started Thu May  7 17:31:51 2020
      1 INFO (TSV-908): verify_test_structures processing complete.

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp TDA-009

-------------------------------------------------------------------------------
```

## Reporting the Test Structures

1. To report the test structure, use command:

```
report_test_structures -workdir mydir -testmode FULLSCAN
```



```
@modus:root:/ 4> report_test_structures -workdir mydir -testmode FULLSCAN
Starting Command: report_test_structures

INFO (TDA-015): Log File: mydir/testresults/logs/log_report_test_structures_FULLSCAN_050720174831-447730000 [end TDA_015]


INFO (TLM-055): Design Summary
                --------------

Hierarchical Model:                  Flattened Model:
           247  Blocks                         212  Blocks
           775  Pins                           212  Nodes
           428  Nets

Primary Inputs:                      Primary Outputs:
          4  Input Only                        9  Output Only
          0  Input/Output                      0  Input/Output
          4  Total Inputs                      9  Total Outputs

Tied Nets:                           Dotted Nets:
         16  Tied to 0                         0  Two-State
          8  Tied to 1                         0  Three-State
          0  Tied to X                         0  Total Dotted Nets
         24  Total Tied Nets

Selected Primitive Functions:
          0  Clock Chopper (CHOP) primitives
          0  RAMs
          0  ROMs
          0  TSDs
          0  Resistors
          0  Transistors
          0  Latches


          8  Rising  Edge Flop w/Set and Reset-Dominant Port
          8  Total Flops

         23  Technology Library Cell Instances

[end TLM_055]

Information for Test Mode:  FULLSCAN
------------------------

        Scan Type = GSD
```

```
Control/Observe Chain Information for Test Mode:  FULLSCAN
-----------------------------------------------

     1 Control Chains (0 Control Only)
     1 Observe Chains (0 Observe Only)
     1 Chains are Control and Observe

  Longest Scan Chain: 8 bits
  Average Scan Chain Length: 8 bits


  Control Chain: 1
    Load Point Pin Index/Name  3/port:scan_in
    Length: 8
    Scan Section: Scan_Section_Sequence
  Observe Chain: 1
    Number of Observe Points: 1
      Observe Point Pin Index/Name          12/port:scan_out    Pipeline Stages: 0
    Unload Point Pin Index/Name  12/port:scan_out
    Unload Point In Phase with Load Point
    Length: 8
    Scan Section: Scan_Section_Sequence


Report completed.


INFO (TDA-001): Maximum Memory used during the run and Cumulative Time in hours:minutes:seconds:

                  Total Memory =            5,547,568  bytes

                      CPU Time =    0:00:00.00
                  Elapsed Time =    0:00:00.17                    [end TDA_001]

     Date Ended:  Thursday May 07 17:48:37 2020  IST




-------------------------------------------------------------------------------
*                      Message Summary                                        *
-------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (TLM-055): Design Summary
```

## Building the Fault Model

1. To build the fault model, use command:

```
build_faultmodel -workdir mydir -fullfault yes
```

```
@modus:root:/ 5> build_faultmodel -workdir mydir -fullfault yes
Starting Command: build_faultmodel

INFO (TDA-015): Log File: mydir/testresults/logs/log_build_faultmodel_050720175406-888111000 [end TDA_015]

INFO (TFM-099): Build Fault Model started.  [end TFM_099]
INFO (TFM-102): Creating faultModel file mydir/tbdata/faultModel.  [end TFM_102]

Global Ignored Static Fault Count           16
Global Ignored Dynamic Fault Count          32


                          Global Statistics

                -- ATCov --   ------------ Global Faults ----------------------
                  Global      Total     Tested   Possibly  Redundant   Untested

Total Static         0.00        398          0          0          0        398
Collapsed Static     0.00        248          0          0          0        248
PI Static            0.00          8          0          0          0          8
PO Static            0.00         18          0          0          0         18

Total Dynamic        0.00        422          0          0          0        422
Collapsed Dynamic    0.00        280          0          0          0        280
PI Dynamic           0.00          8          0          0          0          8
PO Dynamic           0.00         18          0          0          0         18

Parametric

IDDq                 0.00        398          0                              398


INFO (TFM-103): Creating faultStatus file mydir/tbdata/faultStatus.  [end TFM_103]

                          Testmode Statistics: FULLSCAN

                  ---- ATCov ----   ---------------- Testmode Faults -------------------  ------------------ Global Faults -------------------
                Testmode Global      Total     Tested   Possibly  Redundant   Untested      Total     Tested   Possibly  Redundant   Untested

Total Static         0.00   0.00        398          0          0          0        398        398          0          0          0        398
Collapsed Static     0.00   0.00        248          0          0          0        248        248          0          0          0        248
PI Static            0.00   0.00          8          0          0          0          8          8          0          0          0          8
PO Static            0.00   0.00         18          0          0          0         18         18          0          0          0         18
```

```
--------------------------------------------------------------------------------
*                     Message Summary                                          *
--------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (TFM-099): Build Fault Model started.
      1 INFO (TFM-102): Creating faultModel file mydir/tbdata/faultModel.
      1 INFO (TFM-103): Creating faultStatus file mydir/tbdata/faultStatus.
      1 INFO (TFM-109): Build Fault Model has completed with highest level severity message of INFO.
      1 INFO (TFM-704): Maximum Global Test Coverage Statistics:

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp TDA-009

--------------------------------------------------------------------------------
```

## Creating the Scan Test

1. To create the scan test, use command:

```
create_scanchain_tests -workdir mydir -testmode \
FULLSCAN -experiment scan
```

```
@modus:root:/ 6> create_scanchain_tests -workdir mydir -testmode \
 ==> FULLSCAN -experiment scan
Starting Command: create_scanchain_tests

INFO (TDA-015): Log File: mydir/testresults/logs/log_create_scanchain_tests_FULLSCAN_scan_050720184225-718875000 [end TDA_015]



********************************************************************************
Coverage Definitions:
 #Faults  : Number of Active Faults (observable).
 #Tested  : Number of Active Faults marked tested.
 #Possibly: Number of Active Faults marked possibly tested
           (good value is 0 or 1; fault value is X).
 #Redund  : Number of Active Faults untestable due to redundancy.
 #Untested: Number of Active Faults untested.
 %TCov  (%Test Coverage)                    : #Tested / #Faults
 %ATCov (%Adjusted TCov)                    : #Tested / (#Faults-#Redund)
********************************************************************************
********************************************************************************
                          Testmode Statistics: FULLSCAN

                        #Faults    #Tested  #Possibly    #Redund  #Untested  %TCov %ATCov
Total Static                398          0          0          0        398   0.00   0.00
Total Dynamic               338          0          0          0        338   0.00   0.00

                           Global Statistics

                        #Faults    #Tested  #Possibly    #Redund  #Untested  %TCov %ATCov
Total Static                398          0          0          0        398   0.00   0.00
Total Dynamic               422          0          0          0        422   0.00   0.00
********************************************************************************

*************************************************************************************************************
INFO (TTC-110): Starting Scan Test generation                                                    [end TTC_110]

INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -  [end TDA_220]
INFO (TDA-220):    Sim.    Eff.  Detected    Tmode    Global     Untested                        [end TDA_220]
INFO (TDA-220):      1       1        224    56.28%   56.28%          174           00:01.00    [end TDA_220]
INFO (TTC-110): Ending Scan Test generation                                                      [end TTC_110]
*************************************************************************************************************

********************************************************************************
                          Testmode Statistics: FULLSCAN

                        #Faults    #Tested  #Possibly    #Redund  #Untested  %TCov %ATCov
Total Static                398        227          0          0        171  57.04  57.04
Total Dynamic               338        134          0          0        204  39.64  39.64
```

```
              ----Final Pattern Statistics----

 Test Section Type                     # Test Sequences
-------------------------------------------------------
 Scan                                  1
-------------------------------------------------------
 Total                                 1

(I) File(s) generated (bytes and name):

          69632 mydir/tbdata/faultStatus.FULLSCAN.scan
           3882 mydir/tbdata/TBDbin.FULLSCAN.scan

INFO (TDA-001): Maximum Memory used during the run and Cumulative Time in hours:minutes:seconds:

                  Total Memory =            6,127,552  bytes

                      CPU Time =    0:00:00.08
                  Elapsed Time =    0:00:01.18                   [end TDA_001]

     Date Ended:  Thursday May 07 18:42:27 2020  IST



-------------------------------------------------------------------------------
*                     Message Summary                                         *
-------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      3 INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -
      2 INFO (TTC-110): Starting Scan Test generation

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp TDA-009

-------------------------------------------------------------------------------
```

## Creating the Logic Test

1. To create the logic test, use command:

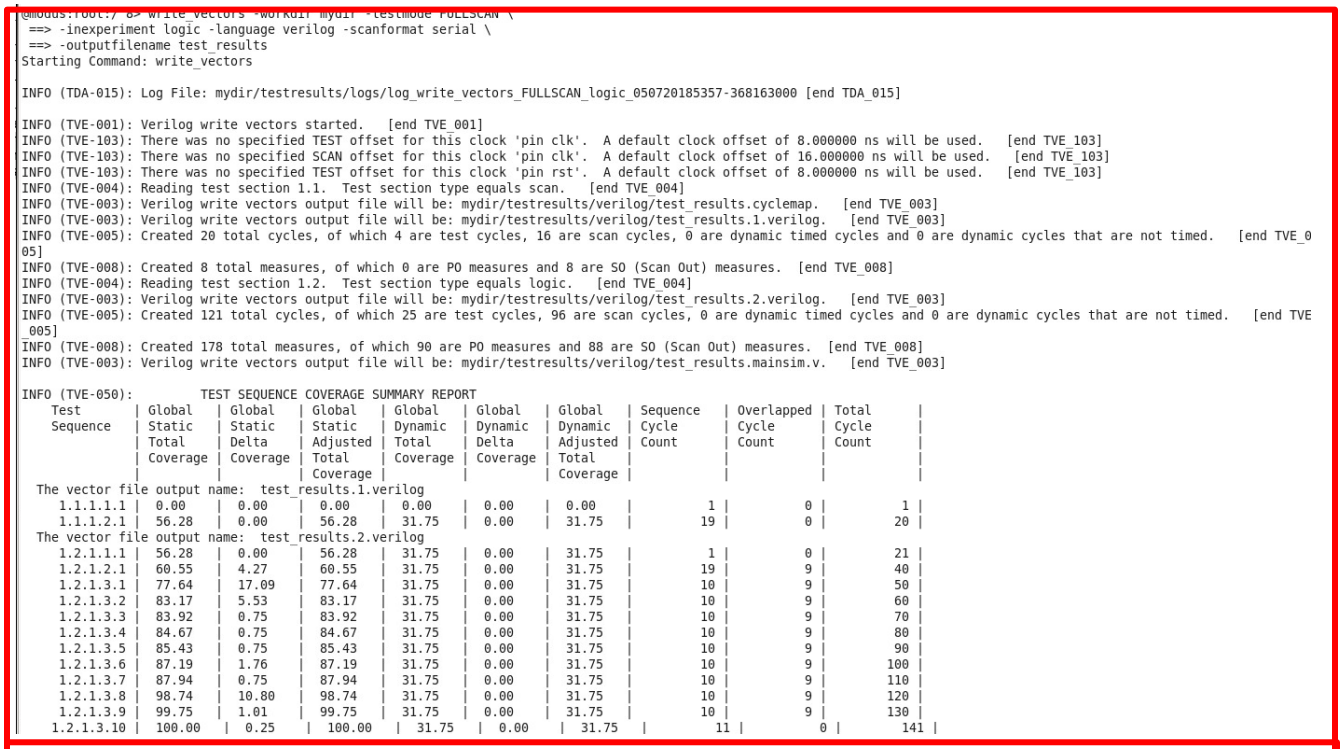
```
create_logic_tests -workdir mydir -testmode FULLSCAN \
   -experiment logic -effort high
```

```
@modus:root:/ 7> create_logic_tests -workdir mydir -testmode FULLSCAN \
 ==> -experiment logic -effort high
Starting Command: create_logic_tests

INFO (TDA-015): Log File: mydir/testresults/logs/log_create_logic_tests_FULLSCAN_logic_050720185014-549199000 [end TDA_015]



********************************************************************************
Coverage Definitions:
 #Faults  : Number of Active Faults (observable).
 #Tested  : Number of Active Faults marked tested.
 #Possibly: Number of Active Faults marked possibly tested
            (good value is 0 or 1; fault value is X).
 #Redund  : Number of Active Faults untestable due to redundancy.
 #Untested: Number of Active Faults untested.
 %TCov  (%Test Coverage)                    : #Tested / #Faults
 %ATCov (%Adjusted TCov)                    : #Tested / (#Faults-#Redund)
********************************************************************************
********************************************************************************
                          Testmode Statistics: FULLSCAN

                        #Faults    #Tested  #Possibly    #Redund  #Untested  %TCov %ATCov
Total Static                398          0          0          0        398   0.00   0.00
Total Dynamic               338          0          0          0        338   0.00   0.00

                           Global Statistics

                        #Faults    #Tested  #Possibly    #Redund  #Untested  %TCov %ATCov
Total Static                398          0          0          0        398   0.00   0.00
Total Dynamic               422          0          0          0        422   0.00   0.00
********************************************************************************

************************************************************************************************************
INFO (TTC-110): Starting Scan Test generation                                                  [end TTC_110]

INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -  [end TDA_220]
INFO (TDA-220):    Sim.    Eff.  Detected    Tmode    Global     Untested                       [end TDA_220]
INFO (TDA-220):       1       1        224    56.28%   56.28%          174           00:00.82    [end TDA_220]
INFO (TTC-110): Ending Scan Test generation                                                    [end TTC_110]
************************************************************************************************************

************************************************************************************************************
INFO (TTC-110): Starting Reset/Set Test Generation                                             [end TTC_110]

INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -  [end TDA_220]
INFO (TDA-220):    Sim.    Eff.  Detected    Tmode    Global     Untested                       [end TDA_220]
INFO (TDA-220):       1       1         17    60.55%   60.55%          157           00:01.08    [end TDA_220]
INFO (TTC-110): Ending Reset/Set Test Generation                                               [end TTC_110]
************************************************************************************************************
```

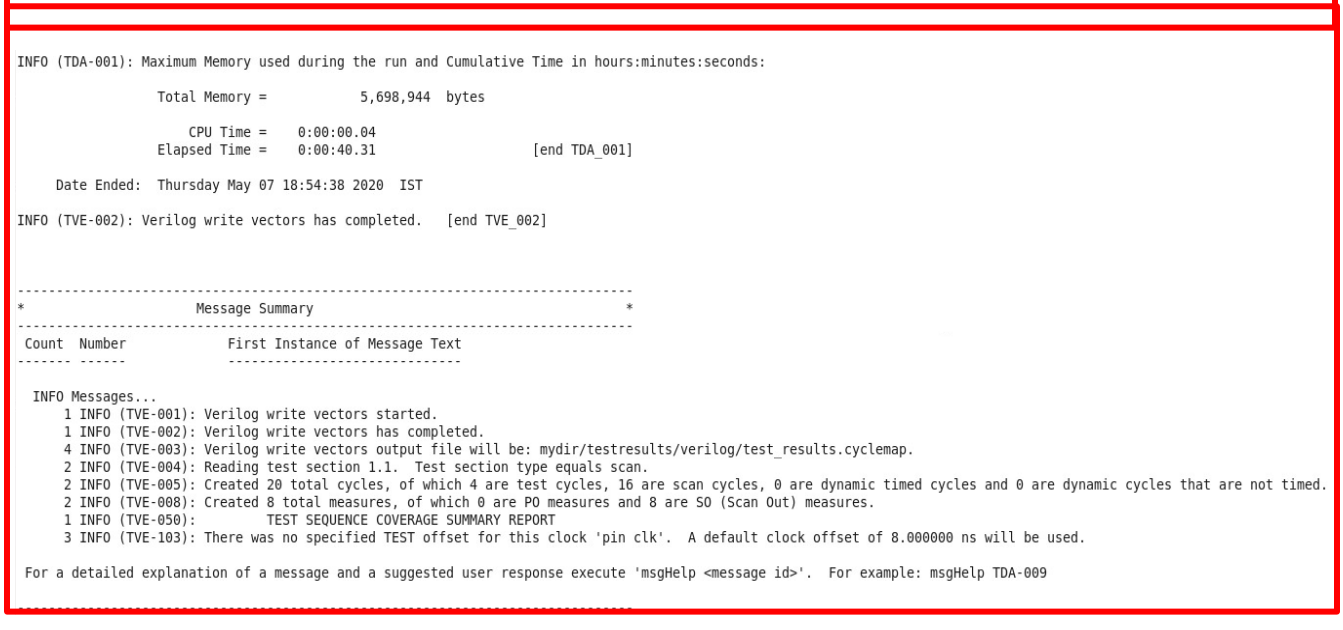
## Writing Out the Vectors

1. To write out the vectors, use command:

```
write_vectors -workdir mydir -testmode FULLSCAN \
   -inexperiment logic -language verilog -scanformat serial \
-outputfilename test_results
```



```
@modus:root:) 8> write_vectors -workdir mydir -testmode FULLSCAN \
 ==> -inexperiment logic -language verilog -scanformat serial \
 ==> -outputfilename test_results
Starting Command: write_vectors

INFO (TDA-015): Log File: mydir/testresults/logs/log_write_vectors_FULLSCAN_logic_050720185357-368163000 [end TDA_015]

INFO (TVE-001): Verilog write vectors started.   [end TVE_001]
INFO (TVE-103): There was no specified TEST offset for this clock 'pin clk'.  A default clock offset of 8.000000 ns will be used.   [end TVE_103]
INFO (TVE-103): There was no specified SCAN offset for this clock 'pin clk'.  A default clock offset of 16.000000 ns will be used.   [end TVE_103]
INFO (TVE-103): There was no specified TEST offset for this clock 'pin rst'.  A default clock offset of 8.000000 ns will be used.   [end TVE_103]
INFO (TVE-004): Reading test section 1.1.  Test section type equals scan.   [end TVE_004]
INFO (TVE-003): Verilog write vectors output file will be: mydir/testresults/verilog/test_results.cyclemap.   [end TVE_003]
INFO (TVE-003): Verilog write vectors output file will be: mydir/testresults/verilog/test_results.1.verilog.   [end TVE_003]
INFO (TVE-005): Created 20 total cycles, of which 4 are test cycles, 16 are scan cycles, 0 are dynamic timed cycles and 0 are dynamic cycles that are not timed.   [end TVE_0
05]
INFO (TVE-008): Created 8 total measures, of which 0 are PO measures and 8 are SO (Scan Out) measures.  [end TVE_008]
INFO (TVE-004): Reading test section 1.2.  Test section type equals logic.   [end TVE_004]
INFO (TVE-003): Verilog write vectors output file will be: mydir/testresults/verilog/test_results.2.verilog.   [end TVE_003]
INFO (TVE-005): Created 121 total cycles, of which 25 are test cycles, 96 are scan cycles, 0 are dynamic timed cycles and 0 are dynamic cycles that are not timed.   [end TVE
_005]
INFO (TVE-008): Created 178 total measures, of which 90 are PO measures and 88 are SO (Scan Out) measures.  [end TVE_008]
INFO (TVE-003): Verilog write vectors output file will be: mydir/testresults/verilog/test_results.mainsim.v.   [end TVE_003]

INFO (TVE-050):         TEST SEQUENCE COVERAGE SUMMARY REPORT
    Test       | Global   | Global   | Global   | Global   | Global   | Global   | Sequence   | Overlapped | Total      |
    Sequence   | Static   | Static   | Static   | Dynamic  | Dynamic  | Dynamic  | Cycle      | Cycle      | Cycle      |
               | Total    | Delta    | Adjusted | Total    | Delta    | Adjusted | Count      | Count      | Count      |
               | Coverage | Coverage | Total    | Coverage | Coverage | Total    |            |            |            |
               |          |          | Coverage |          |          | Coverage |            |            |            |
  The vector file output name:  test_results.1.verilog
     1.1.1.1.1 |  0.00    |  0.00    |  0.00    |  0.00    |  0.00    |  0.00    |          1 |          0 |          1 |
     1.1.1.2.1 |  56.28   |  0.00    |  56.28   |  31.75   |  0.00    |  31.75   |         19 |          0 |         20 |
  The vector file output name:  test_results.2.verilog
     1.2.1.1.1 |  56.28   |  0.00    |  56.28   |  31.75   |  0.00    |  31.75   |          1 |          0 |         21 |
     1.2.1.2.1 |  60.55   |  4.27    |  60.55   |  31.75   |  0.00    |  31.75   |         19 |          9 |         40 |
     1.2.1.3.1 |  77.64   |  17.09   |  77.64   |  31.75   |  0.00    |  31.75   |         10 |          9 |         50 |
     1.2.1.3.2 |  83.17   |  5.53    |  83.17   |  31.75   |  0.00    |  31.75   |         10 |          9 |         60 |
     1.2.1.3.3 |  83.92   |  0.75    |  83.92   |  31.75   |  0.00    |  31.75   |         10 |          9 |         70 |
     1.2.1.3.4 |  84.67   |  0.75    |  84.67   |  31.75   |  0.00    |  31.75   |         10 |          9 |         80 |
     1.2.1.3.5 |  85.43   |  0.75    |  85.43   |  31.75   |  0.00    |  31.75   |         10 |          9 |         90 |
     1.2.1.3.6 |  87.19   |  1.76    |  87.19   |  31.75   |  0.00    |  31.75   |         10 |          9 |        100 |
     1.2.1.3.7 |  87.94   |  0.75    |  87.94   |  31.75   |  0.00    |  31.75   |         10 |          9 |        110 |
     1.2.1.3.8 |  98.74   |  10.80   |  98.74   |  31.75   |  0.00    |  31.75   |         10 |          9 |        120 |
     1.2.1.3.9 |  99.75   |  1.01    |  99.75   |  31.75   |  0.00    |  31.75   |         10 |          9 |        130 |
    1.2.1.3.10 |  100.00  |  0.25    |  100.00  |  31.75   |  0.00    |  31.75   |         11 |          0 |        141 |
```



```
INFO (TDA-001): Maximum Memory used during the run and Cumulative Time in hours:minutes:seconds:

                Total Memory =            5,698,944  bytes

                    CPU Time =    0:00:00.04
                Elapsed Time =    0:00:40.31                   [end TDA_001]

    Date Ended:  Thursday May 07 18:54:38 2020  IST

INFO (TVE-002): Verilog write vectors has completed.   [end TVE_002]



--------------------------------------------------------------------------------
*                     Message Summary                                          *
--------------------------------------------------------------------------------
 Count  Number            First Instance of Message Text
------- ------            ------------------------------

  INFO Messages...
      1 INFO (TVE-001): Verilog write vectors started.
      1 INFO (TVE-002): Verilog write vectors has completed.
      4 INFO (TVE-003): Verilog write vectors output file will be: mydir/testresults/verilog/test_results.cyclemap.
      2 INFO (TVE-004): Reading test section 1.1.  Test section type equals scan.
      2 INFO (TVE-005): Created 20 total cycles, of which 4 are test cycles, 16 are scan cycles, 0 are dynamic timed cycles and 0 are dynamic cycles that are not timed.
      2 INFO (TVE-008): Created 8 total measures, of which 0 are PO measures and 8 are SO (Scan Out) measures.
      1 INFO (TVE-050):         TEST SEQUENCE COVERAGE SUMMARY REPORT
      3 INFO (TVE-103): There was no specified TEST offset for this clock 'pin clk'.  A default clock offset of 8.000000 ns will be used.

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp TDA-009

--------------------------------------------------------------------------------
```

### Exiting the Software

1. To close the software, enter the following command:

```
exit
```

# Module 7: The Equivalency Checking Stage

## Lab 7-1 Running the Equivalence Checking Flow in Conformal

**Objective: To run the basic logic equivalence checking flow for a design.**

The tool used for equivalence checking (comparing RTL to gate-level netlist) is the Conformal® Logic Equivalence Checker (LEC).

Conformal LEC is a tool used for formal verification of designs at various stages in the flow. Formal verification is the process of verifying designs using mathematical methods. Equivalence Checking is the process of verifying the correctness of a modified or transformed design (revised design) by comparing it with a reference design (golden design).

This lab uses the following software release:

- CONFRML232 (23.20-p100)

### Steps to Invoke Tools

1. Before invoking any tool, invoke the C shell by entering “csh” in the terminal.

2. Source the *cshrc* file by entering “source <cshrc file>”, for example:

```
source cshrc
```

### Invoking Equivalency Checking

Change the directory to Equivalence_checking.

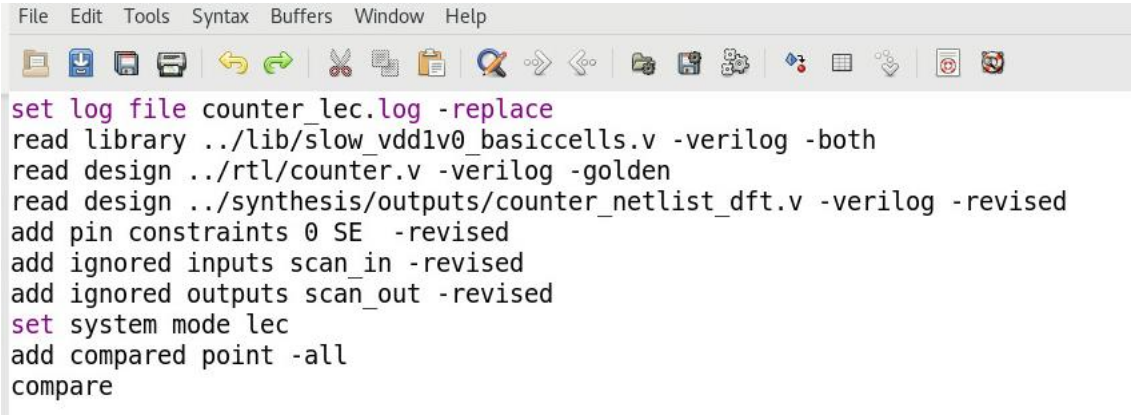
Invoke Conformal LEC inside the Equivalence_checking directory in non-GUI by using the command:

```
lec –XL –nogui -color -64 –dofile <filename>
```

- **-XL**: Launches Conformal L with Datapath and advanced equivalence checking capabilities.
- **-nogui**: Starts the session in non-GUI mode.
- **-color**: Turns on color-coded messaging when in non-GUI mode.
- **-64**: Runs the Conformal software in 64-bit mode.
- **-dofile <filename>**: Runs the script <filename> after starting LEC.

## Creating the Script (DOFILE) for Run

Let us understand the content of the *dofile*. *dofile* is a script file used to run LEC. Shown below is an example of the *dofile*.




1. Save the log file.

   ```
   set log file <filename.log>  - replace
   ```

   Save the log file and replace any log file existing with the same name.

2. Read the Verilog library by entering:

   ```
   read library <filename> -verilog -both
   ```

   - **-verilog**: Indicates that the library is in Verilog format.
   - **-both**: Uses the same library to model or structure both golden and revised designs.

   **Note:** Both Verilog and Liberty formats can be used, but the Verilog format is preferred. Steps to generate *.v* from *.lib* using Conformal are mentioned at the end of this session.

3. Read the Golden Design (RTL) by entering:

   ```
   read design <filename> -verilog -golden
   ```

   - **-verilog**: Indicates that RTL is coded in Verilog.
   - **-golden**: Inputs the golden design.

4. Read the Revised Design by entering:

   ```
   read design <filename> -verilog -revised
   ```

   - **-verilog**: Indicates that the netlist is in Verilog.
   - **-revised**: Inputs the revised design.

5. Ignore the scan input (scan_in) and scan output (scan_out) pins (as these instances are not available in the golden design, and the primary output key point is the compare point):

   ```
   add ignored inputs scan_in –revised
   add ignored outputs scan_out –revised
   ```

   **Note:** This ignores the scan_in and scan_out pins.

6. Add pin constraints 0 SE. Constraint the scan enable (SE) pin to zero to keep the revised design in functional mode:

   ```
   add pin constraints 0 SE -revised
   ```

   **Important:** The tool keeps the design in functional mode and ignores the *scan_in* pin as a support point. Also, *scan_out* is ignored as a compare point.

7. Change the mode of operation from “setup” to “lec”:

   ```
   set system mode lec
   ```

   **Note:** Conformal LEC has two modes of operation: SETUP and LEC. The setup mode is used to prepare the design to be compared. Any command that affects the way the design is modeled will need to be issued in this mode. The LEC mode is where the designs will get modeled, key points mapped, and where the compare process takes place.

8. Compare the Golden with the Revised netlist:

   ```
   add compared points –all
   compare
   ```

## Running Equivalency Checking

1. Change to the *Equivalence_checking* directory by entering this command:

   ```
   cd Equivalence_checking
   ```

2. Start the software by entering this command:

   ```
   lec –XL –nogui -color  -64
   ```

   You can enter commands interactively in the LEC shell.

   The command shell starts LEC in non-GUI mode.

   **Important:** Use the command shown below to invoke Conformal LEC along with the script file.

   ```
   lec –XL –nogui -color  -64 -dofile counter.do
   ```

***lec*** is the command to invoke Conformal LEC, and the *-dofile* option is used to pass the script to LEC at the time of launching the tool. LEC will execute each command mentioned inside the script file one by one.

**Note:** If the script file is in the current working directory (*Equivalence_checking* directory), we need not have to provide the path for the script.

**Important:** If you are moving ahead by sourcing the complete script in one go, you can skip the following section and continue from the section “Analyzing Results after Comparison.”

**Important:** While performing synthesis, always check the LEC terminal to see whether the tool is reporting any error.

## Loading Libraries and Designs and Comparing Designs

1. Set log file and read library:

```
set log file counter_lec.log -replace
read library ../lib/slow_vdd1v0_basiccells.v -verilog -both
```

2. Load Golden and Revised designs:

```
read design ../rtl/counter.v -verilog -golden
read design ../synthesis/outputs/counter_netlist_dft.v -verilog -
   revised
```

3. Add pin constraints and ignore DFT signals:

```
add pin constraints 0 SE  -revised
add ignored inputs scan_in -revised
add ignored outputs scan_out -revised
```

4. Switch the System mode to LEC:

```
set system mode lec
```

5. Add Compare points and compare the designs:

```
add compared points -all
compare
```

## Analyzing Results After Comparison

Once the compare process is completed, Conformal LEC will print a summary report that tells how many key points are equivalent, non-equivalent, aborted, and not compared.

```
// Command: add compare point -all
// 16 compared points added to compare list
// Command: compare
================================================
Compared points       PO      DFF        Total
------------------------------------------------
Equivalent            8       8          16
================================================
```

1. Generate a verification report by entering:

   ```
   report verification
   ```

   (Reports a table of all violated checklist items.)

2. To turn on the GUI window, enter the following command:

   ```
   set gui on
   ```

   If there is a mapping issue or comparison issue or non-equivalence, use the mapping manager or debug manager or schematic viewer options in LEC to resolve the issue.

## Exiting the Software

1. To close LEC, enter the following command:

   ```
   exit
   ```

   **Important:** The same is the flow to compare the netlist generated at different stages of physical design. Use proper modeling directives and constraints to verify designs.

## Lab 7-2 Creating a *.v* Format File from the *.lib* Format

### Objective: To run LEC to create a *.v* file from the *.lib* file format.

In this lab, you will create the output file in *.v* format from an input file in the *.lib* format file using the Conformal Logic Equivalence Checker.

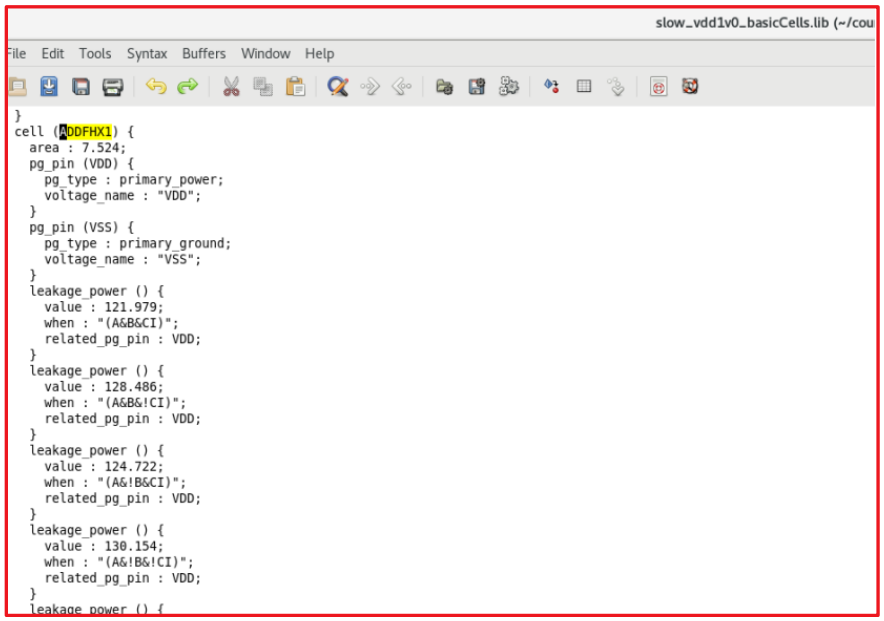
### Understanding the Flow

1. Invoke LEC using the following command:

   ```
   lec -XL -nogui -64
   ```

2. Read the library in liberty (.lib) format using the following command:

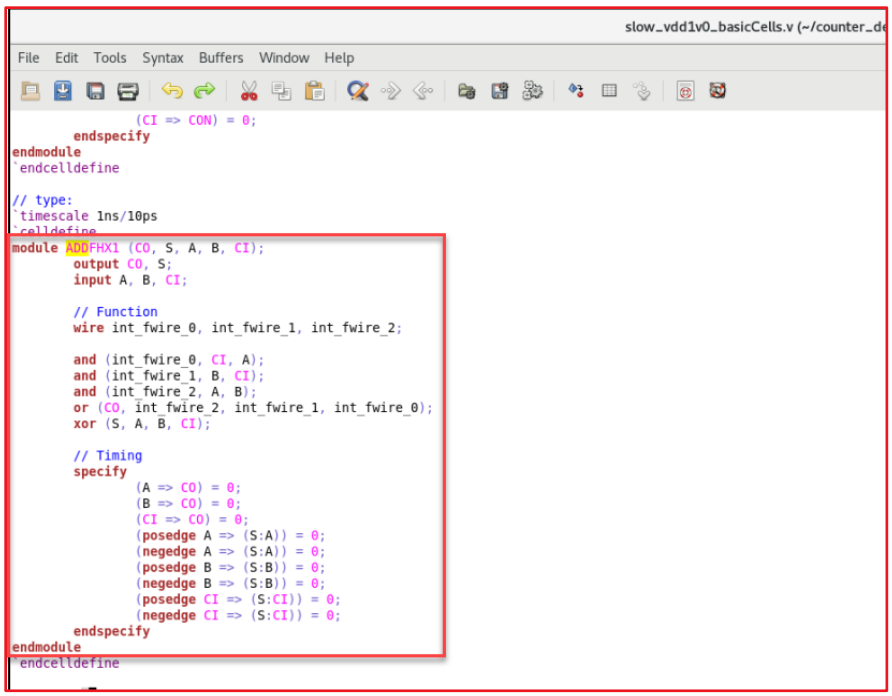
   ```
   read library <.lib file> -liberty -both
   ```




3. Write out the Verilog file using the following command:

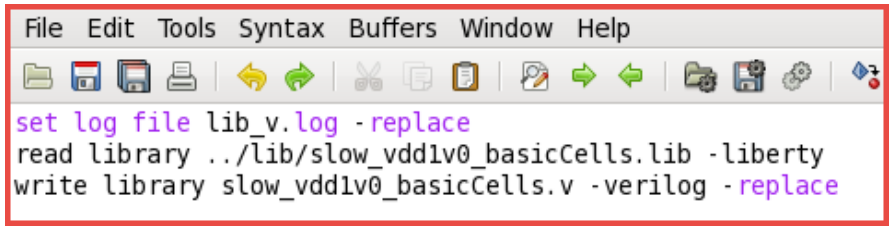
   ```
   write library <file name*> -verilog
   ```

   ***file name** can be any name with the "*.v*" extension.




## Creating a *.v* File

Shown below is an example *dofile* to generate ".v" from ".lib."




1. Set the log file and read the library in *.lib* format:

```
set log file lib_v.log -replace
read library ../lib/slow_vdd1v0_basicCells.lib -liberty -both
```

2. Write out the library in *.v* format:

```
write library slow_vdd1v0_basicCells.v -verilog -replace
```

3. Exit the software:

```
exit
```

End of Lab

# Module 8: The Implementation Stage

# Lab 8-1 Running the Basic Implementation Flow

**Objective: To run the implementation flow including floorplanning, placement, power planning, and routing.**

---

In this lab, you will use the Innovus™ Implementation System to implement the floorplanning, placement, routing, etc., for this design. At the end of the lab, you will also verify your results before handing them off for signoff.

This lab uses the following software release:

- INNOVUS231(23.10-p003_1)

## Importing the Design

In this section, you import a gate-level netlist and libraries into the Innovus Implementation System.

1. Change to the working directory where you will run floorplanning by entering:

   ```
   cd  physical_design
   ```

2. Start the Innovus Implementation System by entering:

   ```
   innovus -stylus
   ```

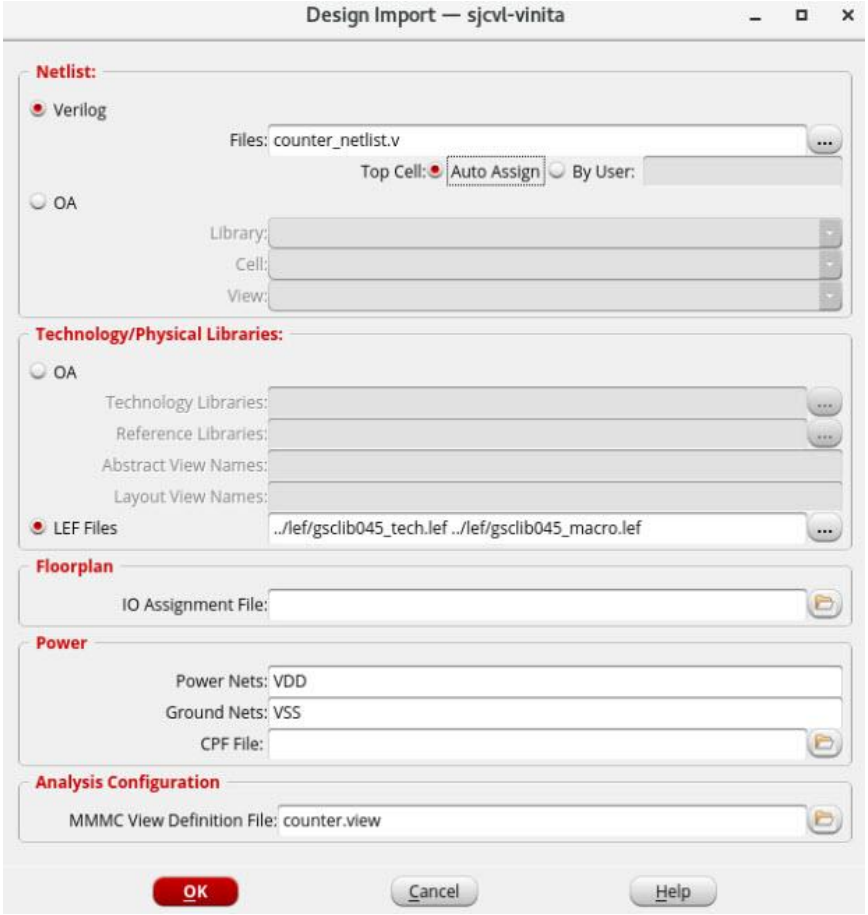
   **Important:** Do not use the window where you started the software for any windowing or UNIX operations except to communicate with the tool.

3. To import a gate-level netlist, timing constraints, and libraries, choose **File – Import Design**.

   The Design Import form appears.

4. Fill in the form as shown.




Here is a brief description of the fields in the Design Import window.

| Field | Description |
|---|---|
| Verilog files | Contain the names of gate-level Verilog netlist files. |
| LEF files | Library of components and physical data for the components in LEF format (also contains routing layers and DRC rules). |
| MMMC View Definition file, *counter.view* | Contains pointers to timing libraries, technology files for extraction, and SDC constraints files. |

5. View the *counter.view* file.

   This file specifies the files for timing analysis and extraction.

```
create_library_set -name max_timing\
   -timing ../lib/slow_vdd1v0_basicCells.lib

create_library_set -name min_timing\
   -timing ../lib/fast_vdd1v0_basicCells.lib

create_timing_condition -name default_mapping_tc_2\
   -library_sets min_timing
create_timing_condition -name default_mapping_tc_1\
   -library_sets max_timing

create_rc_corner -name rccorners\
   -cap_table ../captable/cln28hpl_1p10m+alrdl_5x2yu2yz_typical.capTbl\
   -pre_route_res 1\
   -post_route_res 1\
   -pre_route_cap 1\
   -post_route_cap 1\
   -post_route_cross_cap 1\
   -pre_route_clock_res 0\
   -pre_route_clock_cap 0\
   -qrc_tech ../QRC_Tech/gpdk045.tch

create_delay_corner -name max_delay\
   -timing_condition {default_mapping_tc_1}\
   -rc_corner rccorners
create_delay_corner -name min_delay\
   -timing_condition {default_mapping_tc_2}\
   -rc_corner rccorners

create_constraint_mode -name sdc_cons\
   -sdc_files\
    counter_sdc.sdc

create_analysis_view -name wc -constraint_mode sdc_cons -delay_corner max_delay
create_analysis_view -name bc -constraint_mode sdc_cons -delay_corner min_delay

set_analysis_view -setup wc -hold bc
```

**Note:** The *counter.view* file sets up corners and modes by setting up pointers to the timing library and the constraints files.

6. Click **OK** to Import the design**.**

### Viewing the Design Import Results

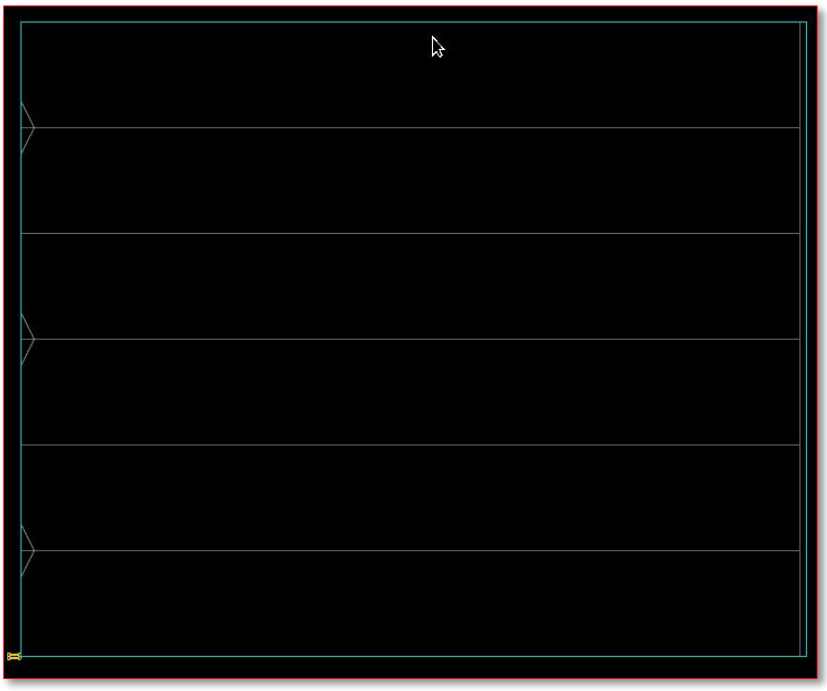
In this section, you learn how to view and interpret what you see in the design window.

1. To enlarge the window, drag the corner of the window until you can see all the modules in your design, as well as all the Innovus menus.

2. Select the **Floorplan** view. 

3. Fit the design to the window by pressing the **f** key.

4. Zoom out by pressing **Shift-Z** or by clicking the **Zoom Out** icon. 



5. Expand the design window to display all the available pull-down menus.

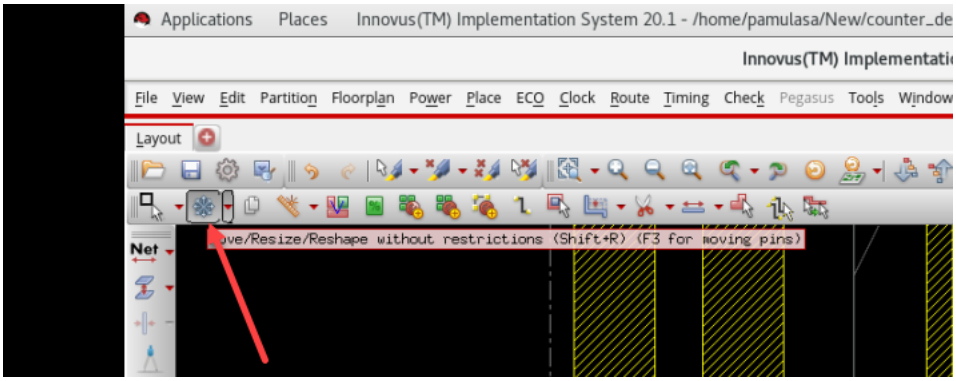
6. To view more of the objects, click the **Zoom Out** icon  or press **Shift-Z**.

7. Move your cursor over the icons and notice that their functions are displayed in text boxes, as shown here.


8. To zoom to a particular area, press and drag the **right** mouse button over a rectangular area.

   The window zooms to that area.

9. To move any object, follow the steps below.




   a. Select the object.

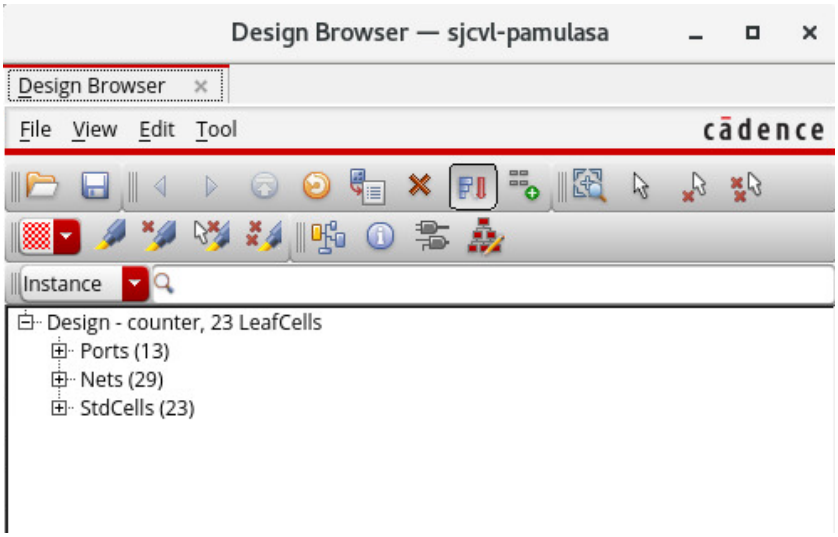
   b. Click on the  icon.

   c. Drag the object to the required position.

## Viewing the Design Hierarchy

1. To view the hierarchical design that you imported, choose **Tools – Design Browser**.

2. To expand the modules, click the plus sign (**+**) next to the categories of design objects.




3. To view the I/O terminals, click the **Ports** plus sign (+).

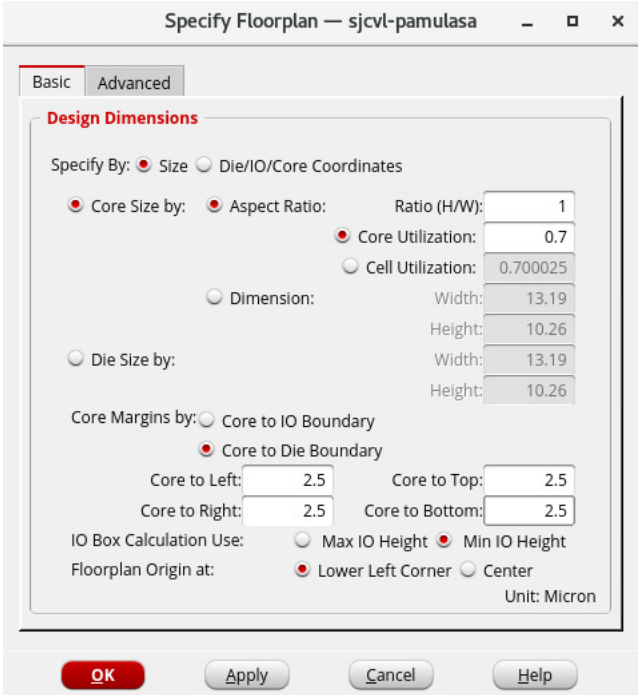
4. When you are finished, close the Design Browser window.

## Floorplanning the Design

In this section, you create a floorplan. You become familiar with the floorplanning forms and icons.

1. Choose **Floorplan – Specify Floorplan**.

   The Specify Floorplan form appears.




   a. For aspect ratio, enter **1**.

   b. Enter Core Utilization **0.7**.

   c. Select **Core to Die Boundary**.

   d. Enter **2.5** for the Core to Left, Core to Right, Core to Top and Core to Bottom values.

      **Note:** To display more information about the options, on the Specify Floorplan form, click **Help**.

e. To initialize the floorplan, click **OK**.

The floorplan you will see should look like the screenshot shown above.

2. To measure the distance between the core area and the I/O boundary, click the **Ruler** icon or press **k**.

3. To delete the ruler, click the **Clear All ruler** icon  or press **Shift+K.**

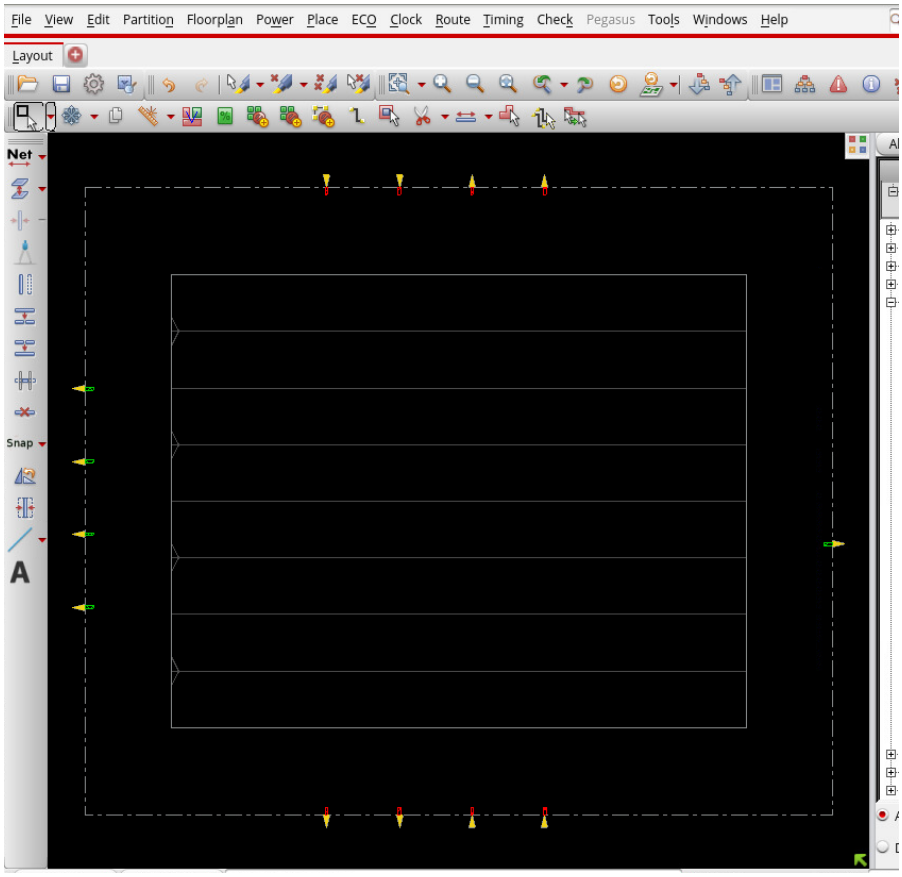
### Pin Assignment

In this section, you will place Input-Output Pins (I/O) pins around the block.

1. Place I/O pins by sourcing the *pins.io* file using the following command.

```
read_io_file pins.io
```

2. Refresh the screen using the redraw  button and view the Pin placement in the design window.



## Power Planning

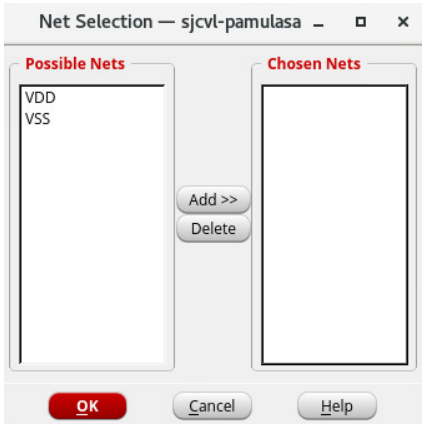
In this section, you create a power plan.

1. Choose **Power-Power Planning – Add Ring**.

   **Note:** The Add Rings form is displayed. There are 11 metal layers available for routing in the horizontal and vertical directions, and you will be selecting some of them for creating rings in subsequent steps.

   a. To select the **VDD** and **VSS** nets, click the folder icon  in the Net(s) field.

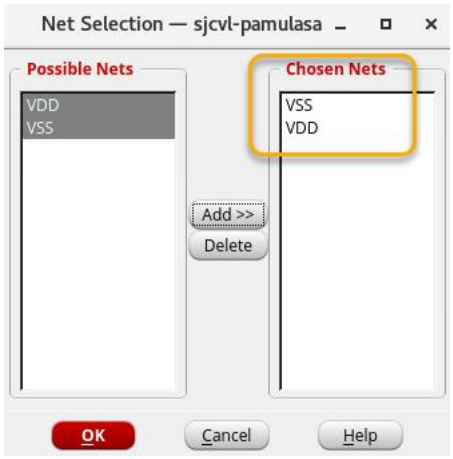
   The Net Selection form is displayed.




   b. In the Possible Nets pane, press **Shift** and **VDD** and **VSS**.

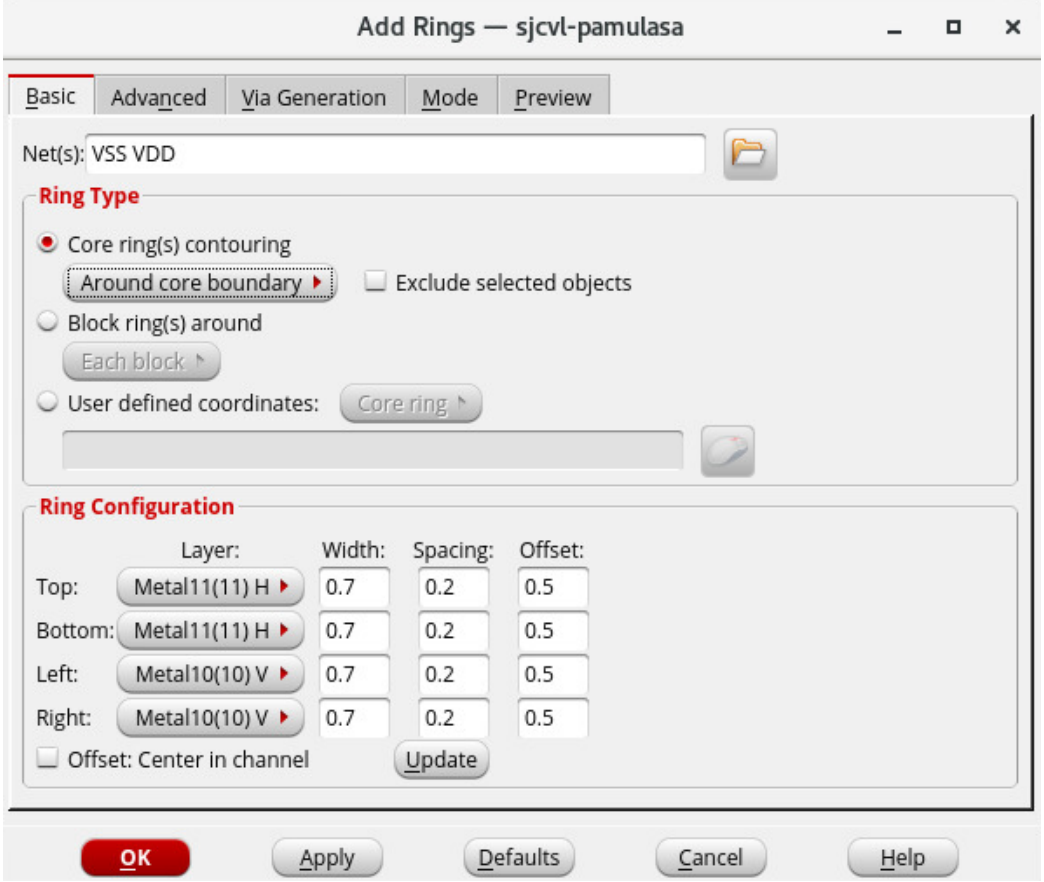
   c. Click **Add**.

   The selected nets appear in the Chosen Nets pane.




   d. Click **OK**.

   e. Make sure that the Net(s) field contains **VDD** and **VSS**.




f. Click the **Core ring(s)** contouring.

g. Select **Around core boundary**.

h. In the Ring Configuration field, make sure that **METAL11 H** layer is selected for Top and Bottom.

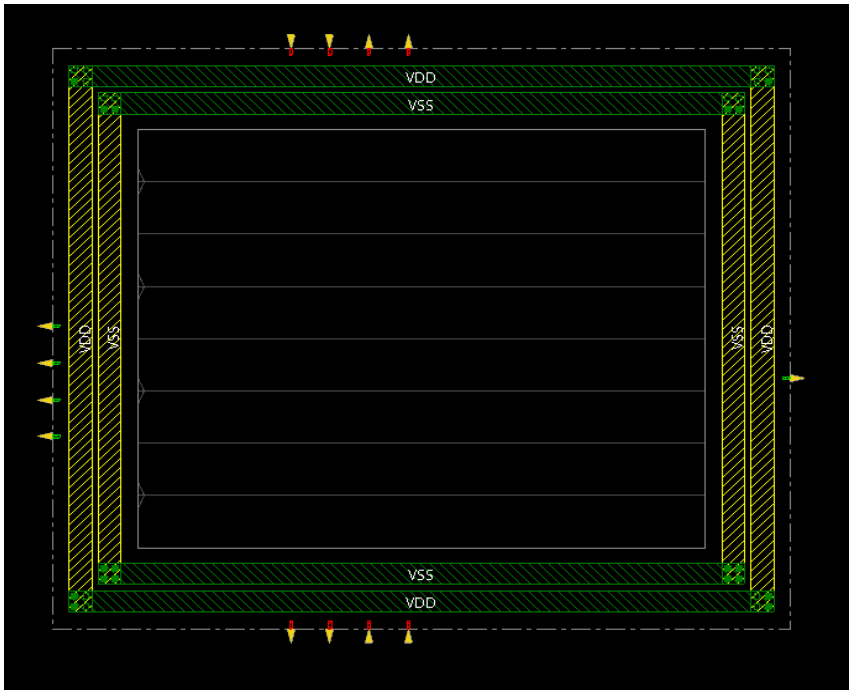
i. Use **METAL10 V** as the layer for Left and Right.

j. For Width, change the default value to **0.7**.

k. For Spacing, change the default value to **0.2**.

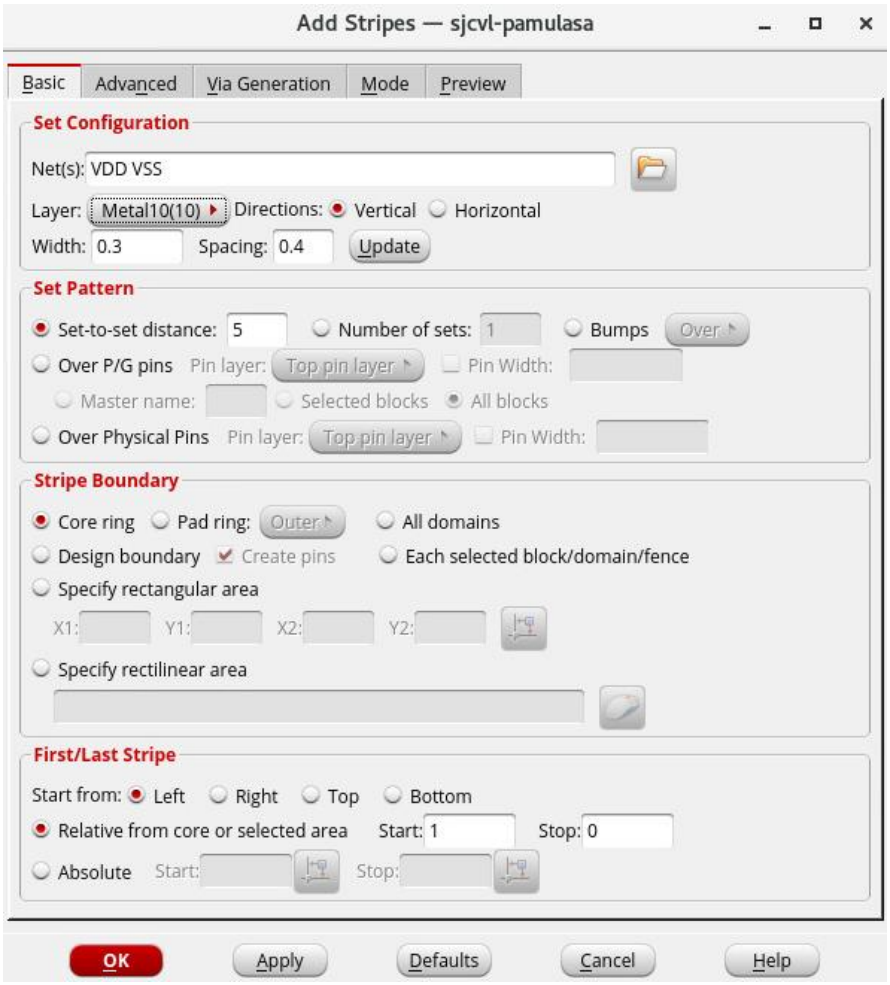
l. For **Offset**, change the default value to **0.5**.

m. To generate the power rings, click **Apply**.



2. Choose **Power – Power Planning** – **Add Stripe**.

   The Add Stripes form appears.




   a. Make sure that the Net(s) field contains **VDD** and **VSS**.

   b. In the cyclic field, select **Metal10**.

   c. Select **Vertical**, if it is not already selected.

   d. Change Width to **0.3**.

   e. Change Spacing to **0.4**.

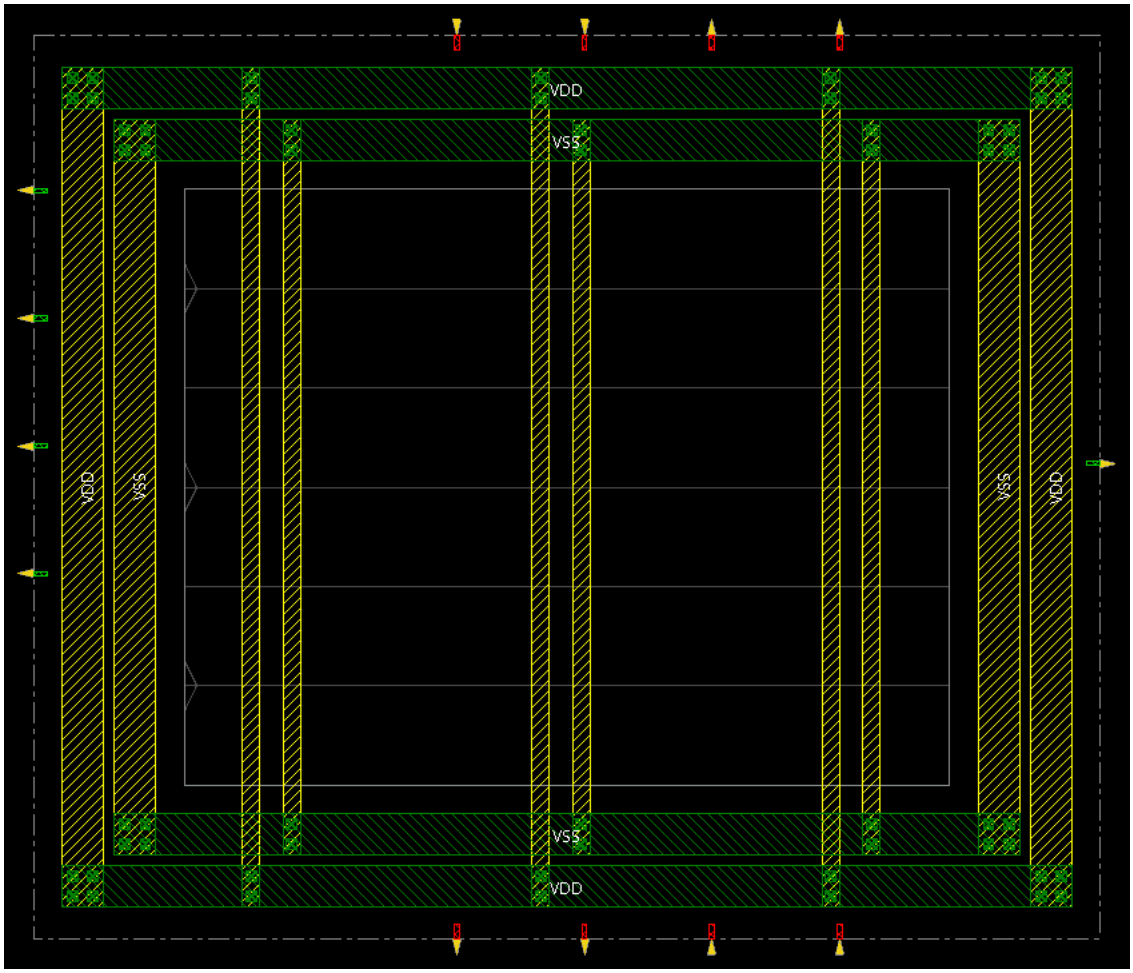
   f. Enter **5** in the *Set-to-set distance* field.

   g. Select **Relative** from core or selected area.

   h. Set Start to **1**.

   i. Set Stop to **0**.

j. Click **OK**.




Notice the power stripes and the vias connecting the rings to the stripes are created.

k. Save the floorplan by selecting **File – Save – Floorplan**.

l. Specify *counter.fp* for the **File Name**.

m. Click **Save.**

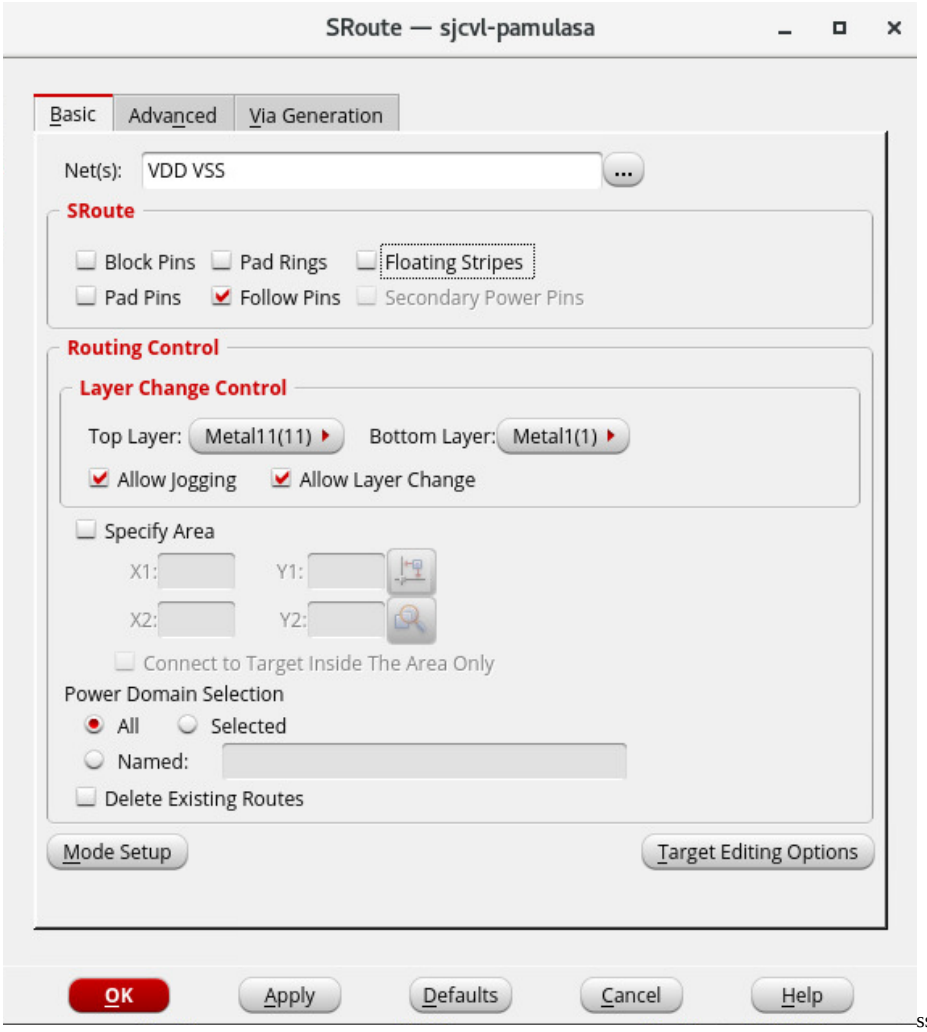
## Creating Power Rails with Special Route

1. Before creating followpin routing (also known as power rails), associate the global VDD and VSS nets names to the standard cell pin names by entering the following commands:

```
connect_global_net VDD -type pg_pin -pin VDD -inst_base_name *
connect_global_net VSS -type pg_pin -pin VSS -inst_base_name *
```

2. Choose **Route – Special Route**.

   a. The SRoute form appears.



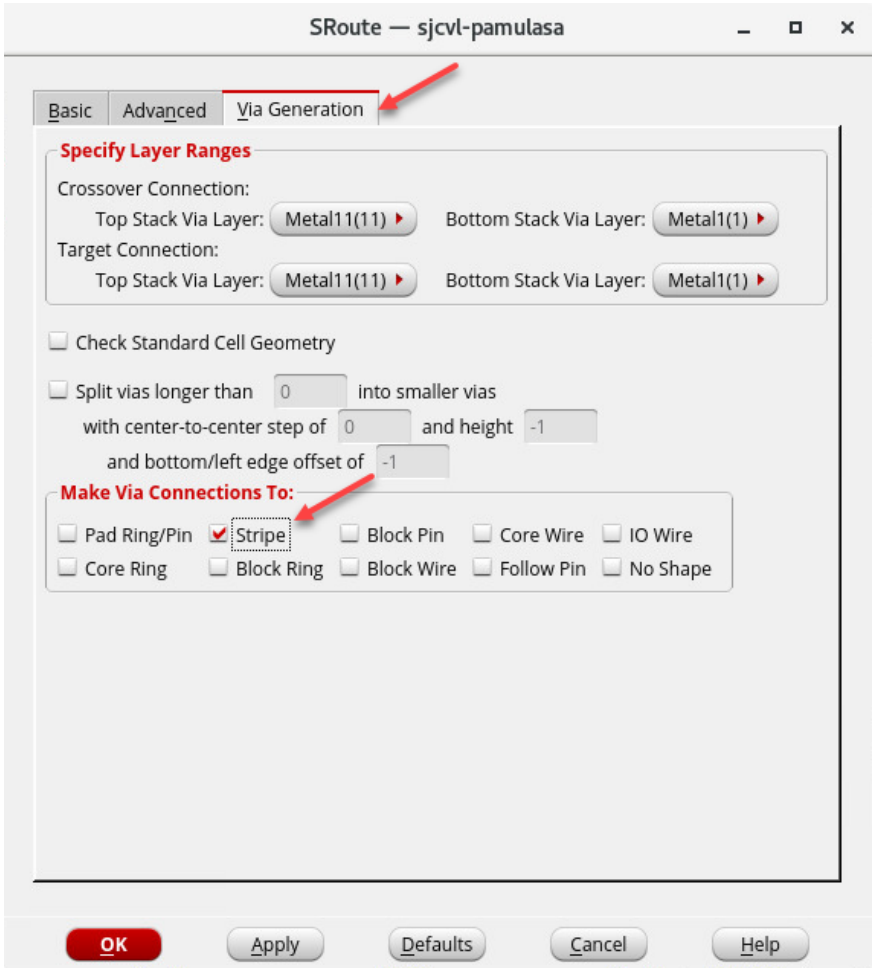
sss

   b. To populate the field with **VDD** and **VSS**, click the icon next to the **Net(s)** field.

   c. Click **Add** to add the nets to the Chosen Nets field and click **OK**.

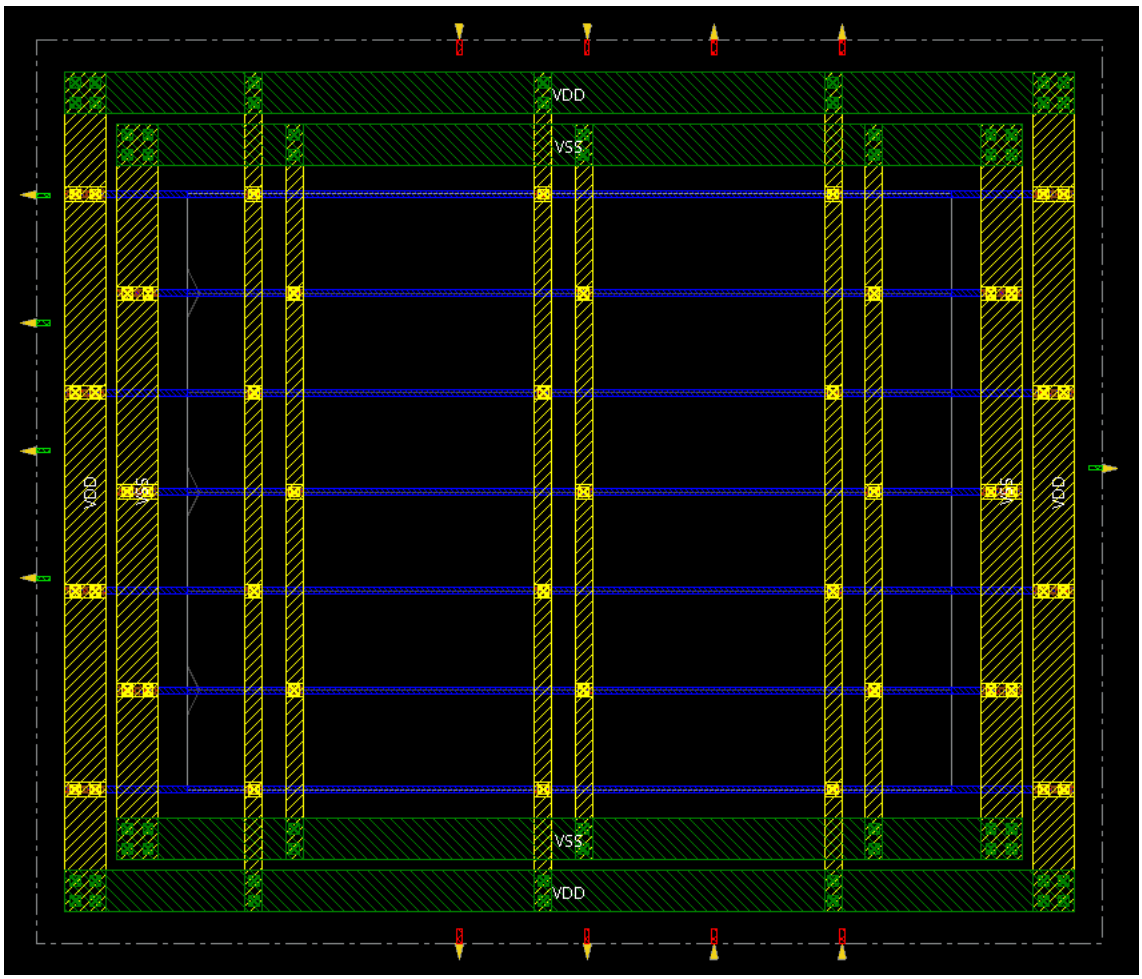
   d. Deselect all options *except* **Follow Pins**.

e. Click Via Generation and Select **Stripe** in the *Make Via Connections To* field.




f. Click **OK**.

3. In the Physical view, zoom in on the followpin routes.




Notice that the power routes have been connected to the power planned targets with relevant vias.

## Running Placement Optimization

1. Load the scan DEF file by running the following command:

   ```
   read_def counter.scandef
   ```

2. Set the scan reordering mode by running the following command:

   ```
   set_db reorder_scan_comp_logic true
   ```

3. Run placement optimization by running the following command:

   ```
   place_opt_design
   ```

   a. The command takes a few minutes to finish. After placement, post-placement setup optimization is run if the Slack is negative.

   b. During the optimization stage of the command, the following operations may be performed to close timing:

   - Adding buffers
   - Resizing gates
   - Reconstructing the circuit
   - Remapping the logic
   - Swapping the pins
   - Deleting the buffers
   - Moving the instances

   **Note:** Notice that the status of the design on the lower-right corner has changed.

   *After the placement run is completed, what is the status of the design that is displayed?*

   Answer: _____

   This field is a convenient way to check where you are in the flow. The timing summary is output to the log file, which contains the Total Negative Slack (TNS) and the Worst Negative Slack (WNS).

   *What is the Worst Negative Slack (WNS) at this stage?*

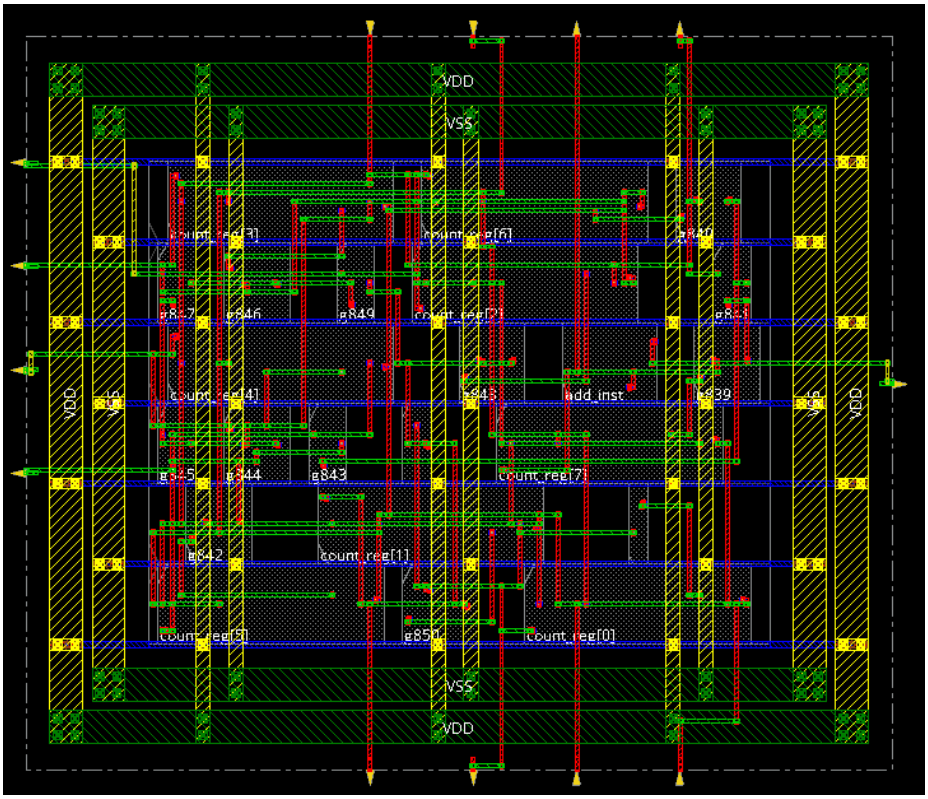
   Answer: _____

   *Is it a positive or a negative number?*

   Answer: _____

4. To display the Physical view, click the **Physical View** button. 

You will see the standard cell placements.




**Note:** Notice that in addition to cell placement, Trial Route has been run on the design.

5. To save the design, enter:

```
write_db placeOpt
```

After running placement or pre-CTS optimization, you run clock tree synthesis with constraints on what buffers to use and the type of clock routing to implement.

## Running Clock Tree Synthesis

1. Generate the clock tree spec file constraints from the *.sdc* file by running the following command:

   ```
   create_clock_tree_spec
   ```

2. Create a clock tree by running the following command:

   ```
   clock_opt_design
   ```

   a. You will see an error message about the clock net not being completely routed.

   **Note:** Ignore this error, as later on, when you run the NanoRoute™ tool for the remaining nets, this error will be fixed.

3. View the *.log* file for this session.

   *Were there any timing violations after this step?*

   Answer: _____

4. Save your design as *postCTSopt* by entering:

   ```
   write_db postCTSopt
   ```

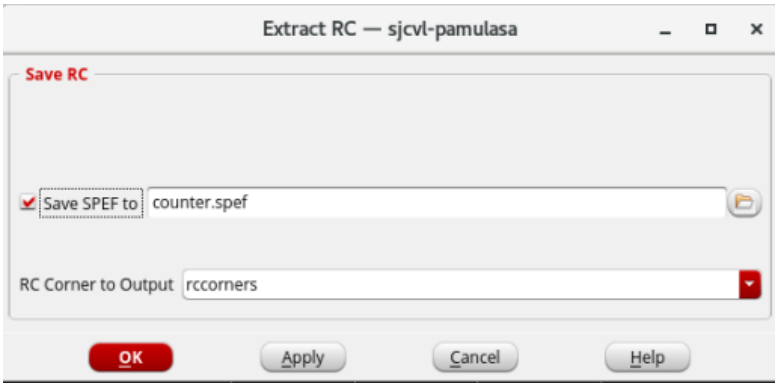
## Routing the Nets

1. To route the nets, choose **Route – NanoRoute – Route**.
   a. The NanoRoute form is displayed.


   b. Make sure that **Timing Driven** is selected.
   c. Select **SI Driven**.
   d. Click **OK in the NanoRoute** form.

## Extraction and Timing Analysis

1. Run RC extraction on the routed design by selecting **Timing – Extract RC**.




2. Enable the check box option **Save SPEF to**

3. Click **OK**.

4. Set the timing analysis mode by running the following commands:

```
set_db timing_analysis_type ocv
```

5. Run setup-and-hold timing analysis by running the following commands:

```
time_design -post_route
time_design -post_route -hold
```

*Are there any setup violations?*

Answer: _____

*Are there any Hold violations?*

Answer: _____

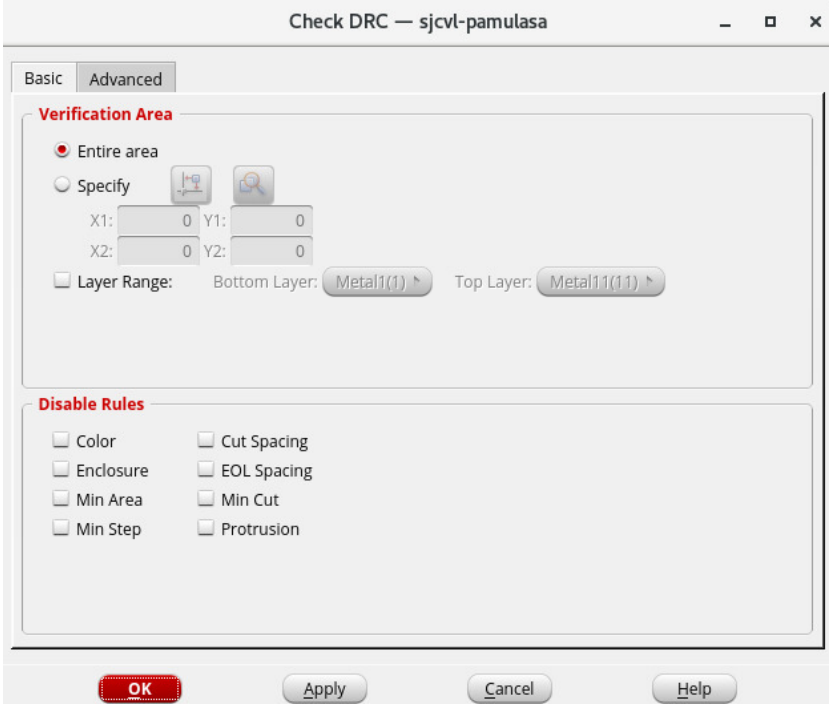
## Running Physical Verification

In this section, you run physical verification commands in the Innovus™ system.

## Verifying Geometry

1. Choose **Check – Check DRC.**

2. Click **OK** with the default options selected in the **Check DRC** window.



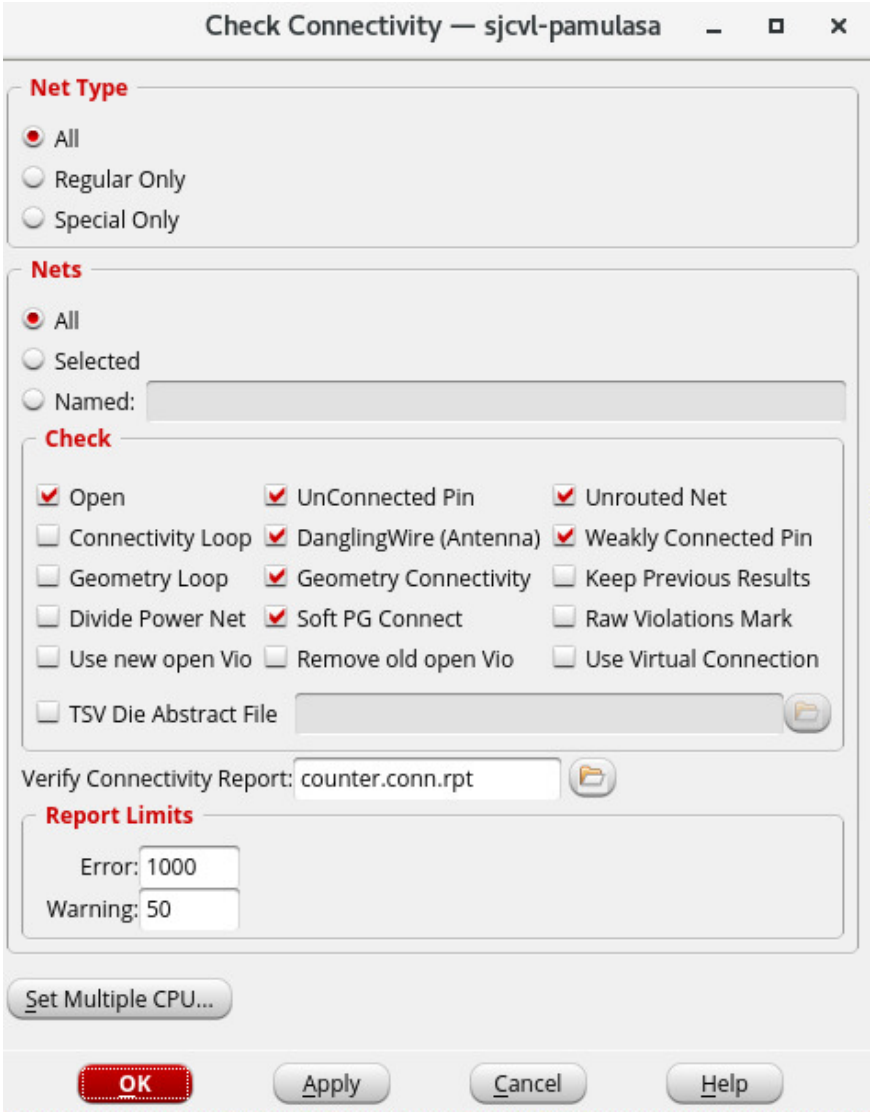

*Are there any violations?*

Answer: _____

## Verifying Connectivity

1. Choose **Check – Check Connectivity**.

2. Click **OK** with the default options selected in the **Check Connectivity** window.

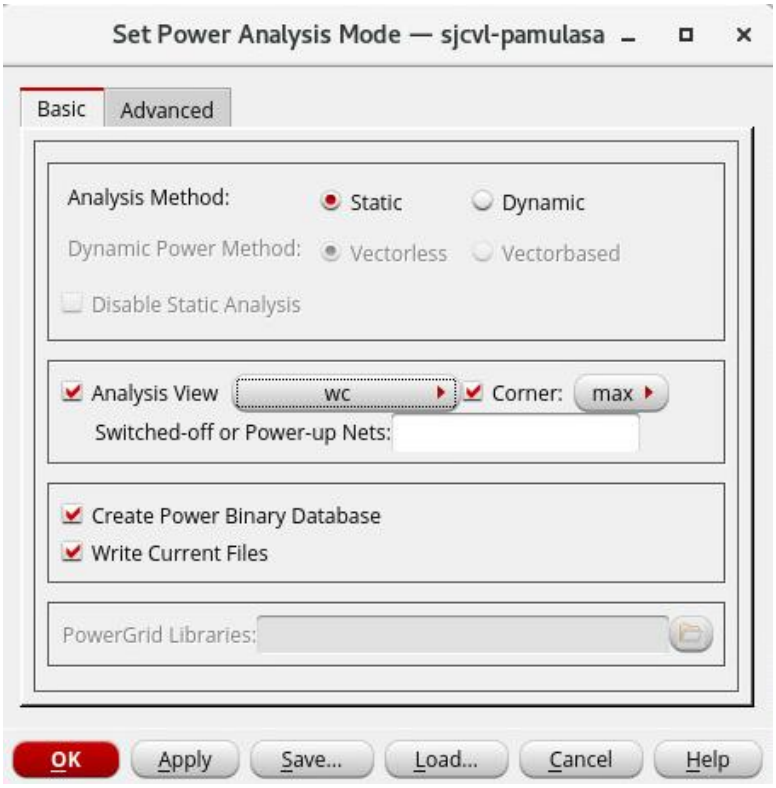



*Are there any violations?*

Answer: _____

## Running Power Analysis

1. To display the Power Analysis Setup form, choose **Power – Power Analysis – Setup.**




   a. Select **Static** for Analysis Method.

   b. Select **Analysis View**.

   c. Select **wc**.

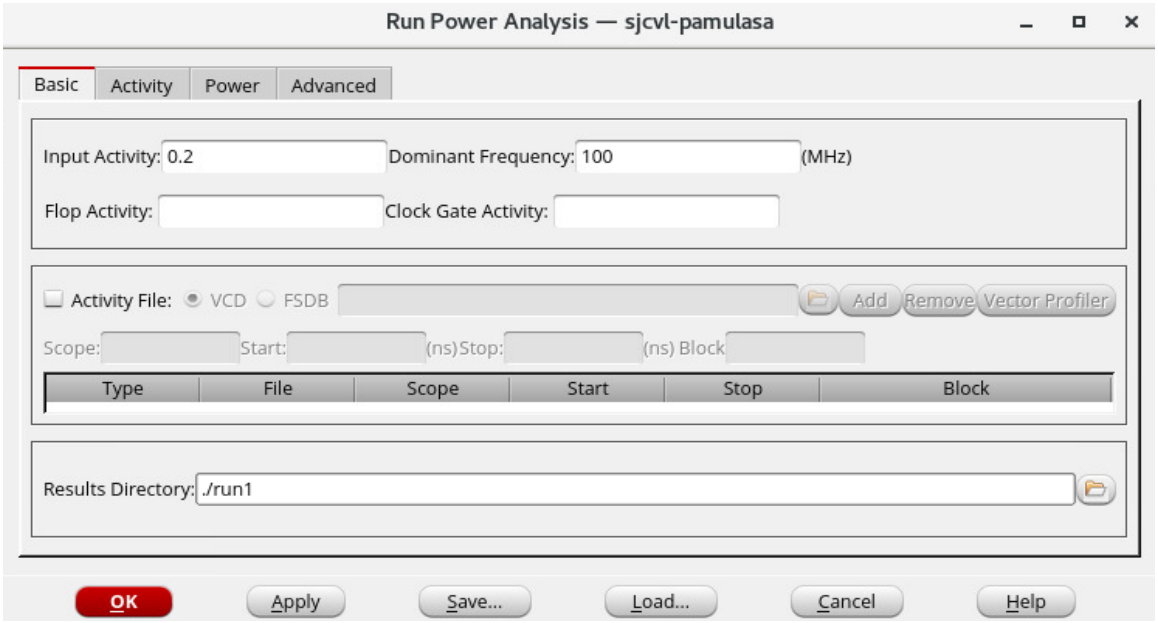
   d. Select **max** for the corner.

   e. Click **OK**.

2. Source the *power.tcl* file, which contains the rules for global net power connections by entering the following:

```
source power.tcl
```

3. To run power analysis, select **Power – Power Analysis – Run.**




4. Leaving all defaults as it is, enter **./run1** in the Results Directory field.

5. Click **OK**.

   This will run a power analysis and generate power consumption values.

6. From the log file for this session, determine the following:

   *What is the total internal power?*

   Answer: _____

   *What is the total switching power?*

   Answer:

   *What is the leakage power?*

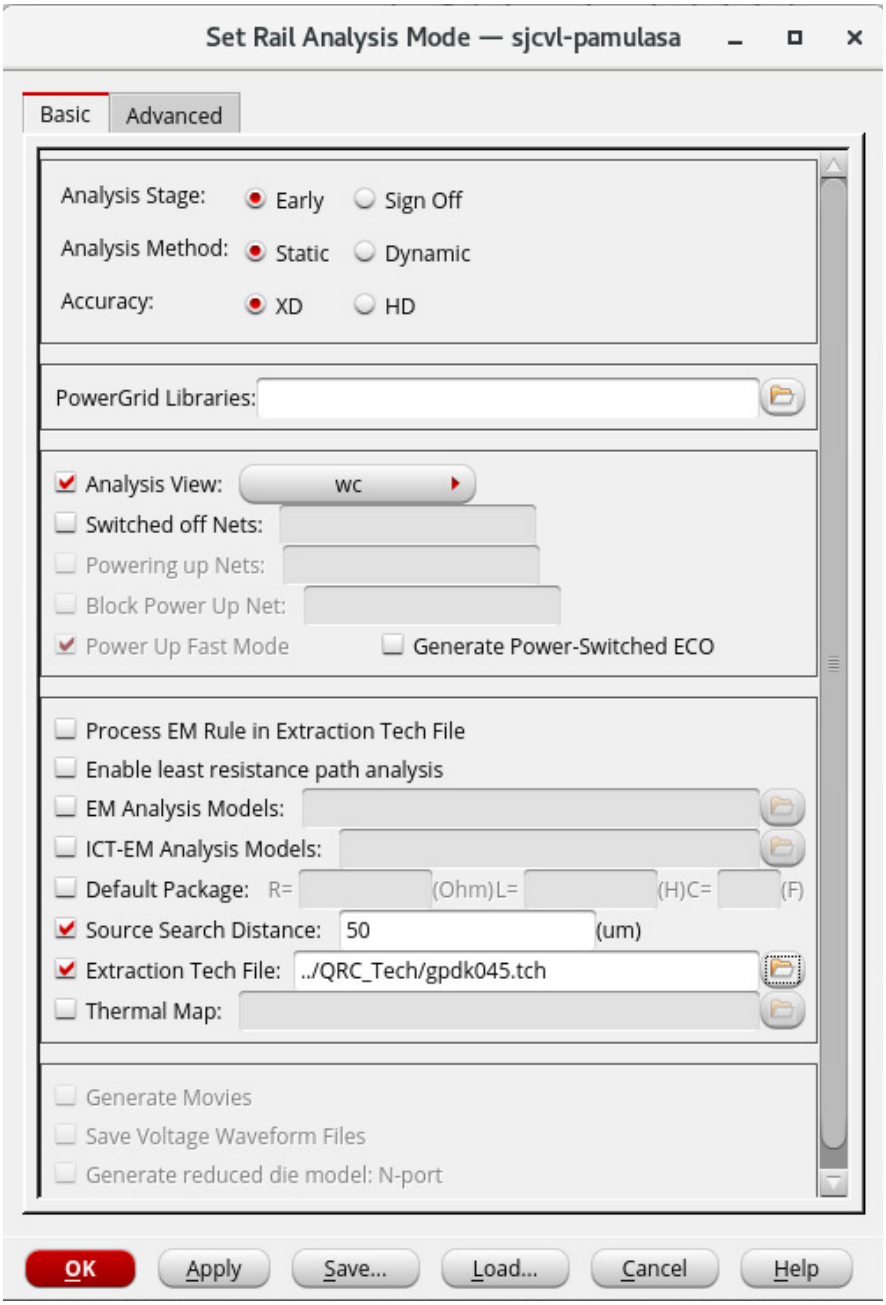
   Answer:

   *What is the total power consumed?*

   Answer: _____

7. To run rail analysis, first, select **Power – Rail Analysis – Setup.**

   a. This will bring up the Set Rail Analysis Mode form.




b. Select **Early** for Analysis Stage.

c. Select **Analysis View**.

d. Select **wc** for the worst case.

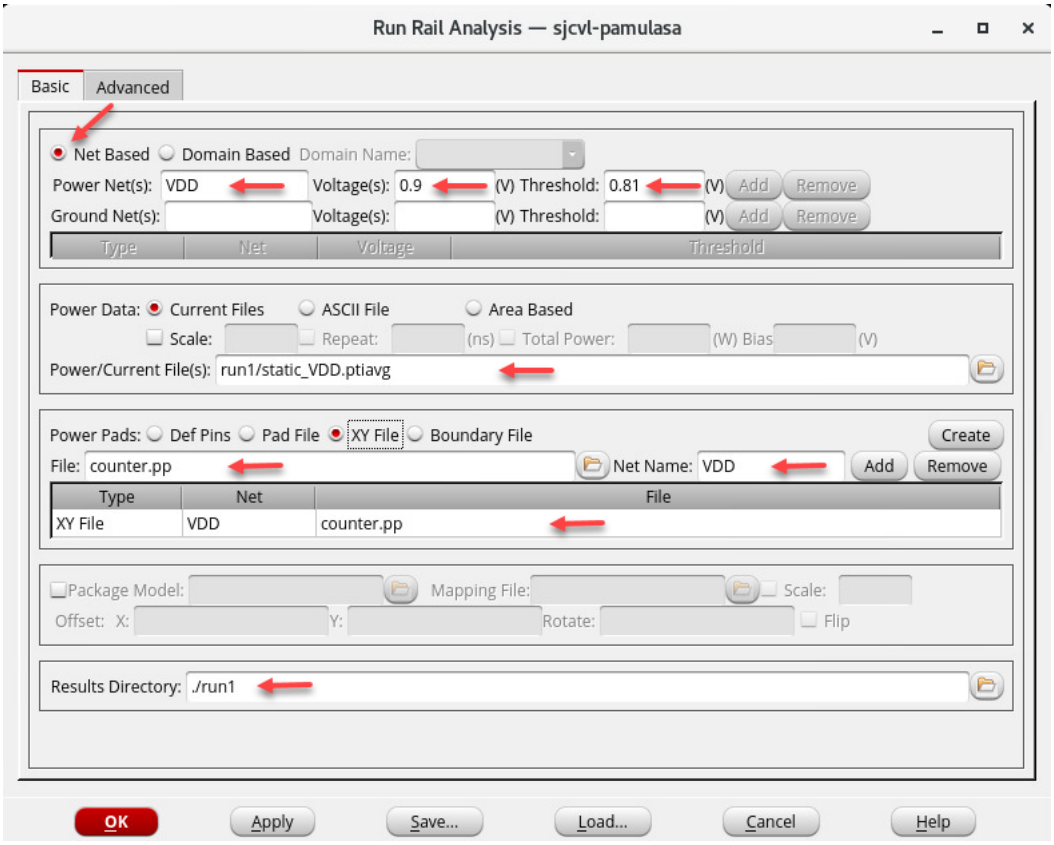
e. Select **Extraction Tech File** and enter **../QRC_Tech/gpdk045tch**.

   Go to the **QRC_Tech** file and choose the filters field with All files(*) and select **gpdk045tch**.

f. Click **OK**.

8. Select **Power – Rail Analysis – Run**.

   This will bring up the Run Rail Analysis form.




   a. In the Run Rail Analysis form:

   b. Select **Net Based**.

   c. Enter **VDD** for the Power Net.

   d. Delete **VSS** if it is in the Ground Net field.

   e. Enter **0.9** in the Voltage(s) field.

   f. Enter **0.81** in the Threshold field.

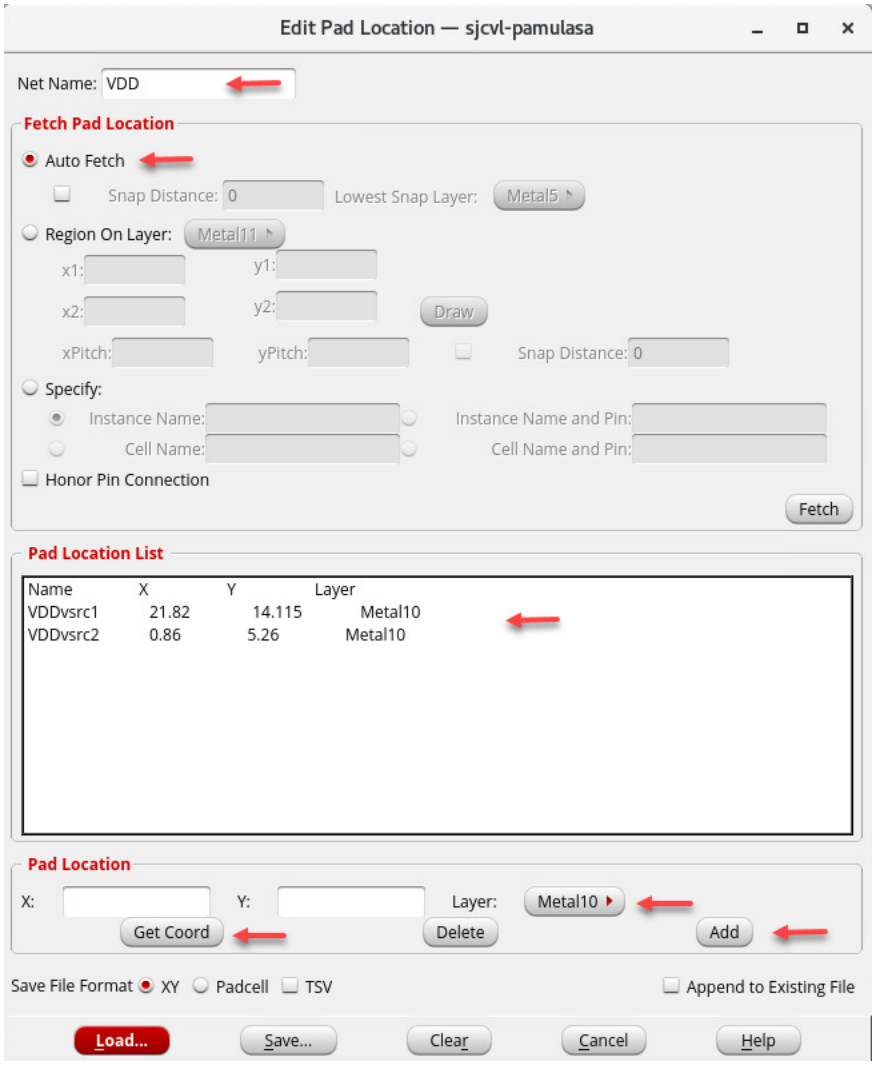
   g. Make sure that **Current Files** is selected for Power Data.

   h. For **Power/Current** Files, enter **run1/static_VDD.ptiavg**.

   i. To create an XY file for the power sources, select **XY** File.

9. Click **Create**.

This will bring up the Edit Pad Location form.




10. In the Edit Pad Location form:
    a. Enter **VDD** for the Net Name.
    b. Make sure **Auto Fetch** is selected.
    c. At the bottom of the Edit Pad Location form, click **Get Coord**.
    d. The cursor will change into a crosshair.
    e. Click on a location in the main Innovus design window.
    f. Go back to the *Edit Pad Location* form.
    g. Make sure to change the Layer to **Metal10** (Metal11 is selected by default).
    h. Click **Add**.

**Note:** Notice that the Pad Location List field is populated with the coordinate as VDDvsrc1.

i. Add another location for a VDD source.

j. Click **Save**.

k. Enter **Counter.pp** for the file name and Click **Save**.

l. Click **Cancel** to close the *Edit Pad Location* form.

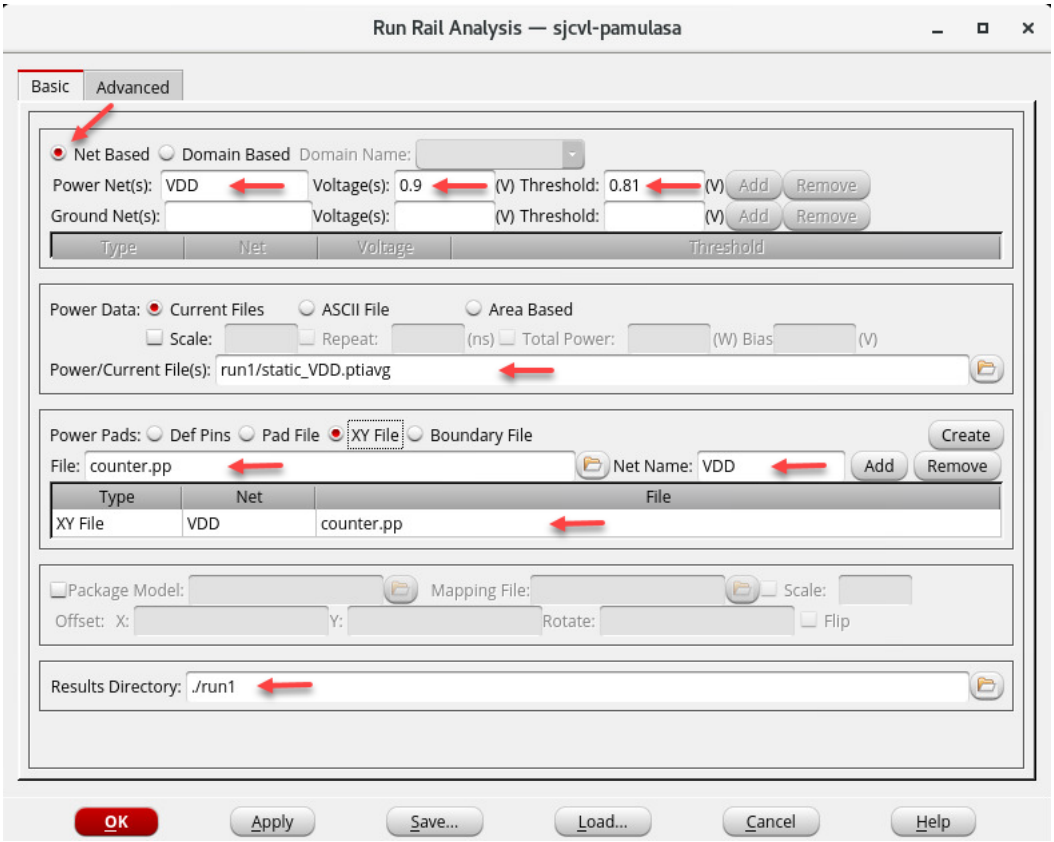
11. In the Run Rail Analysis form, specify the name of the XY file (*Counter.pp*) you just created in the File field.

12. Enter **VDD** for the Net Name.

13. Click **Add** so that the pane under the fields is populated with the type of file, net name and file name.

14. Enter **./run1** for the Results directory.

15. Make sure that the form looks like what is shown below.



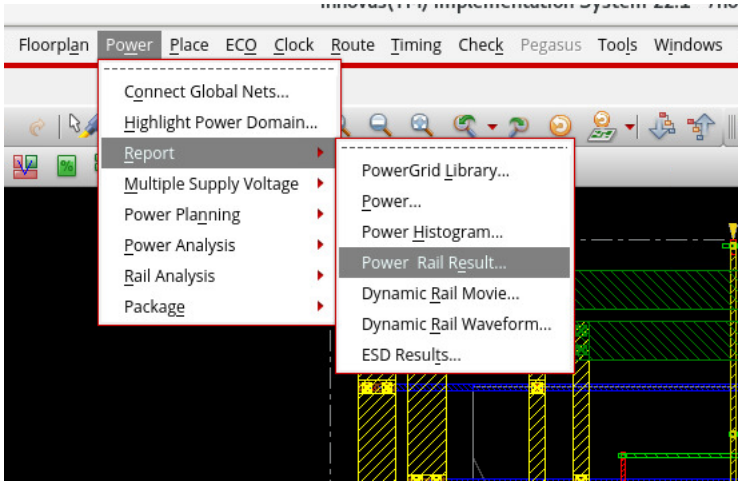
16. Click **OK** to run rail analysis.

## Viewing Power Analysis Results

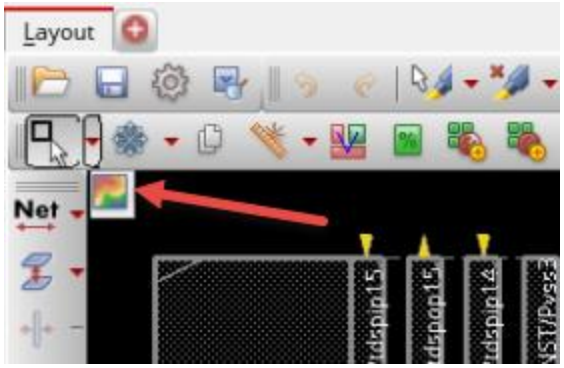
1. To display the results of rail analysis, load the power and rail databases.

```
read_power_rail_results -power_db run1/power.db -rail_directory
   run1/VDD_25C_avg_1
```

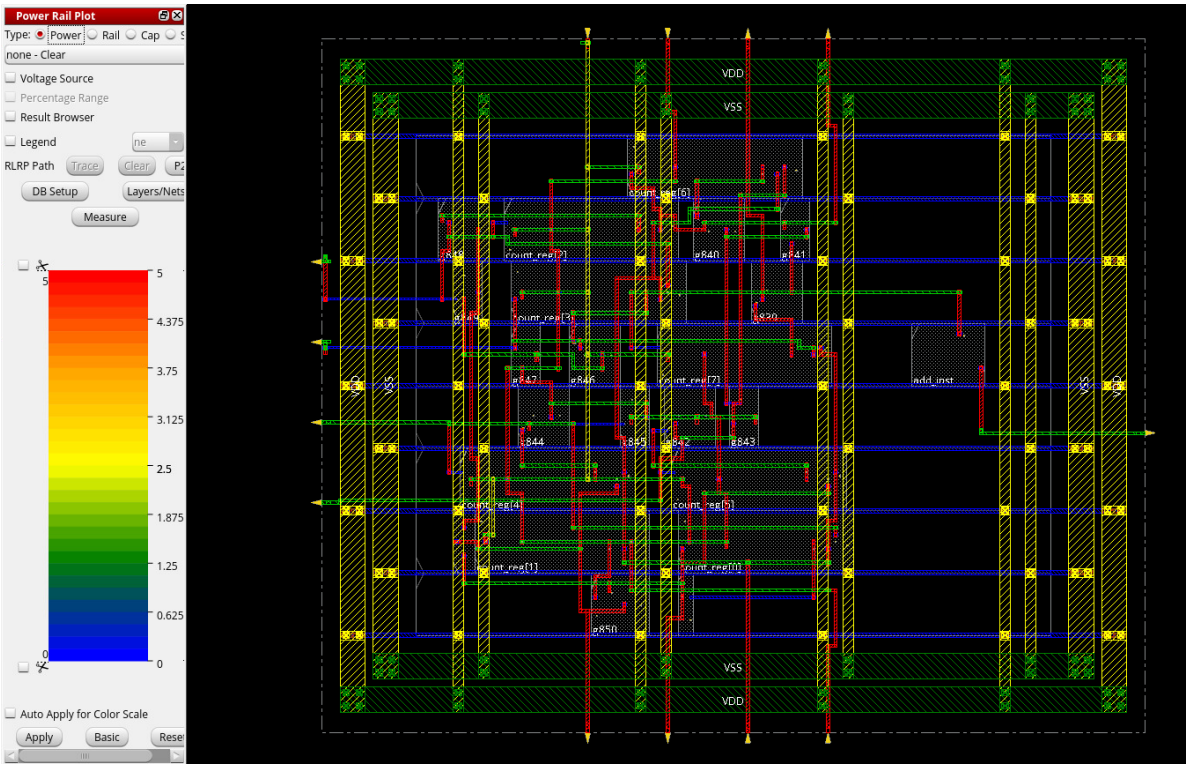
2. Select **Power – Report - Power Rail Result.**



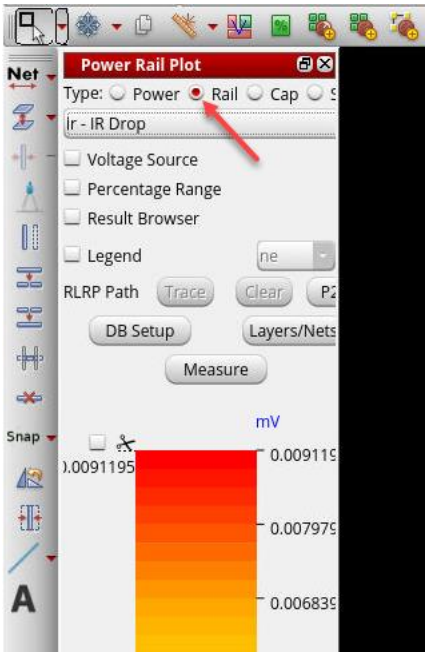

3. **Double**-click the icon to bring up the Power Rail Plot.



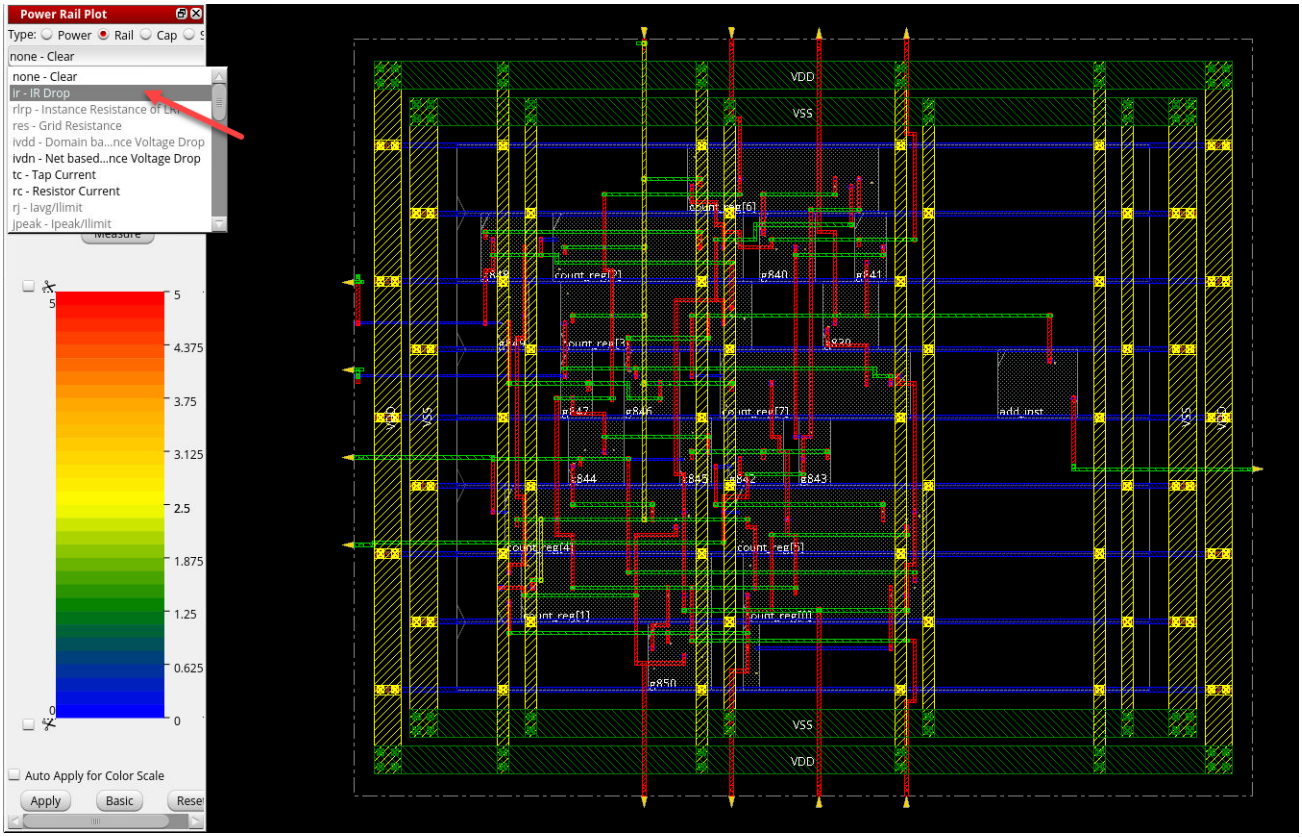
4. Notice that the pane on the left of the design window contains options to display the results.




5. Select **Rail**.



6. Select **ir – IR Drop** from the pick list.




**Note:** Notice that the Innovus design window display shows the color-coded voltage ranges that match the range.

*Are there any red areas displayed in the main Innovus window?*

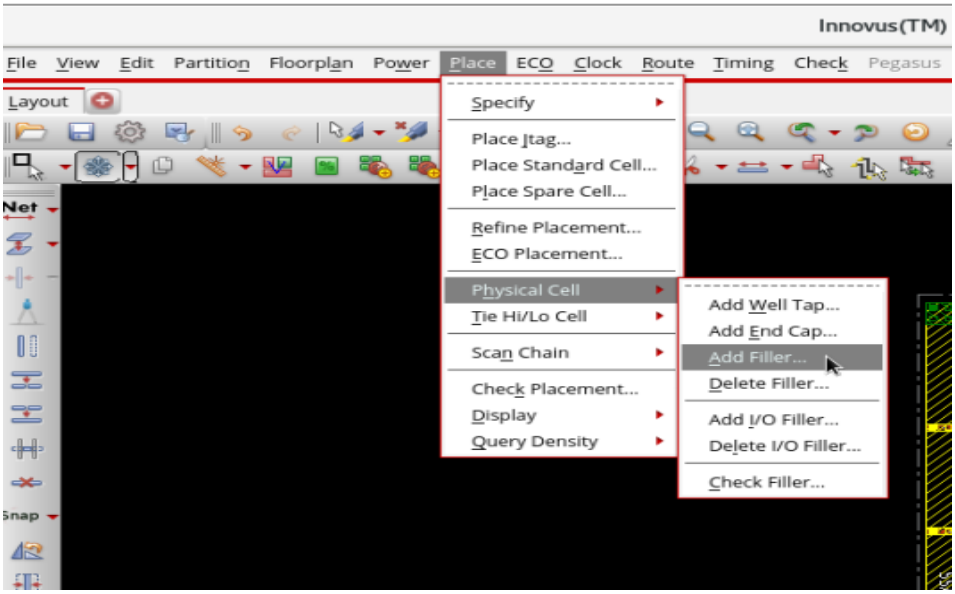
Answer: _____

7. Enter the following to save the Innovus database:

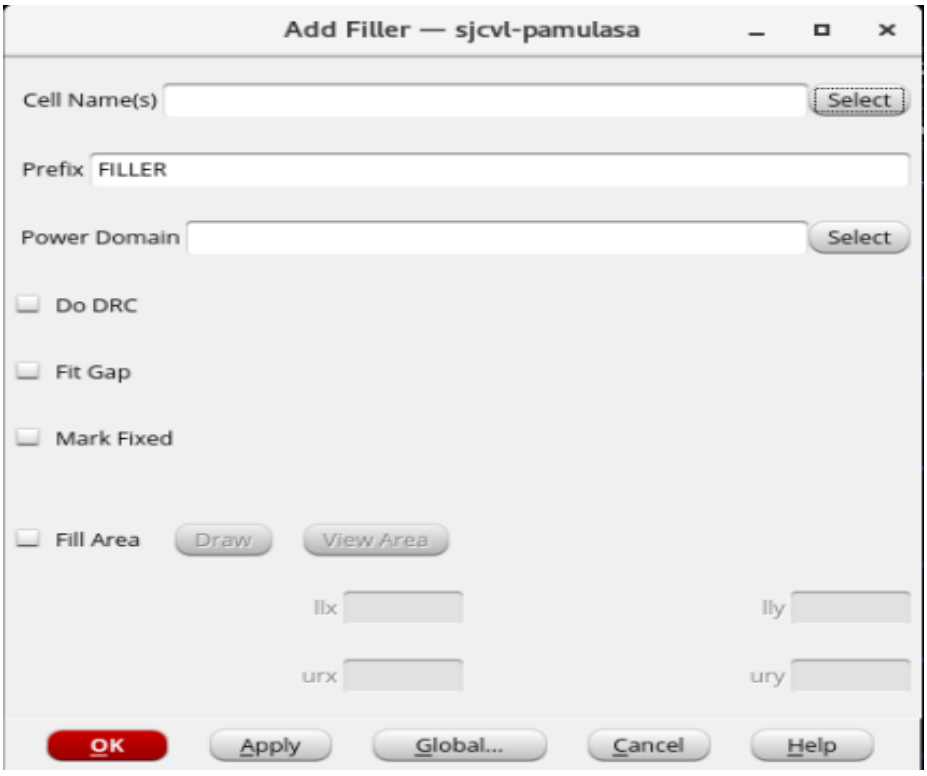
```
write_db counter.inn -lib
```

## Filler Cell Placement

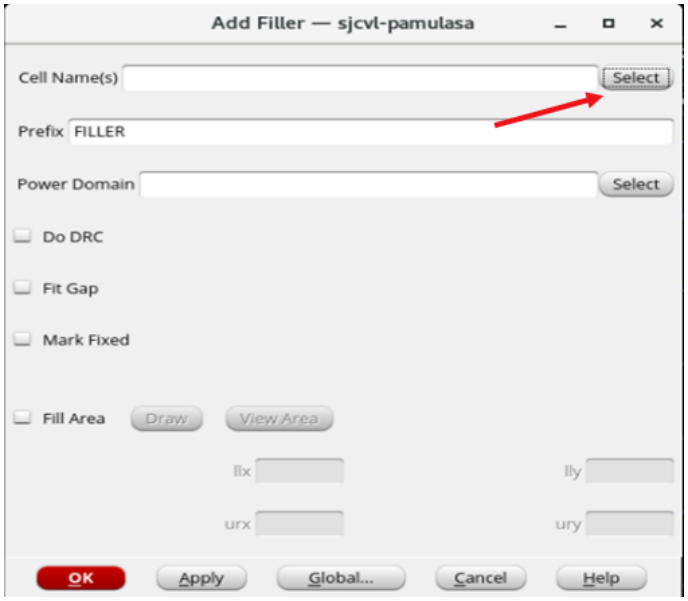
1. Select **Place-Physical Cell-Add Filler**.



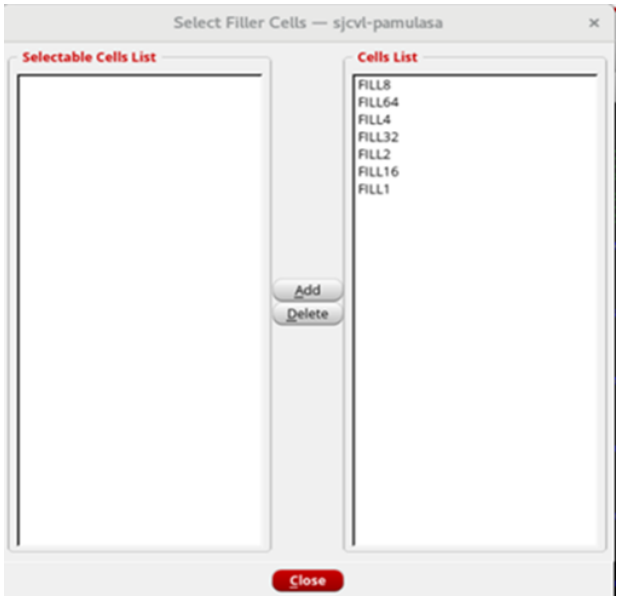

This will bring up an Add filler form.



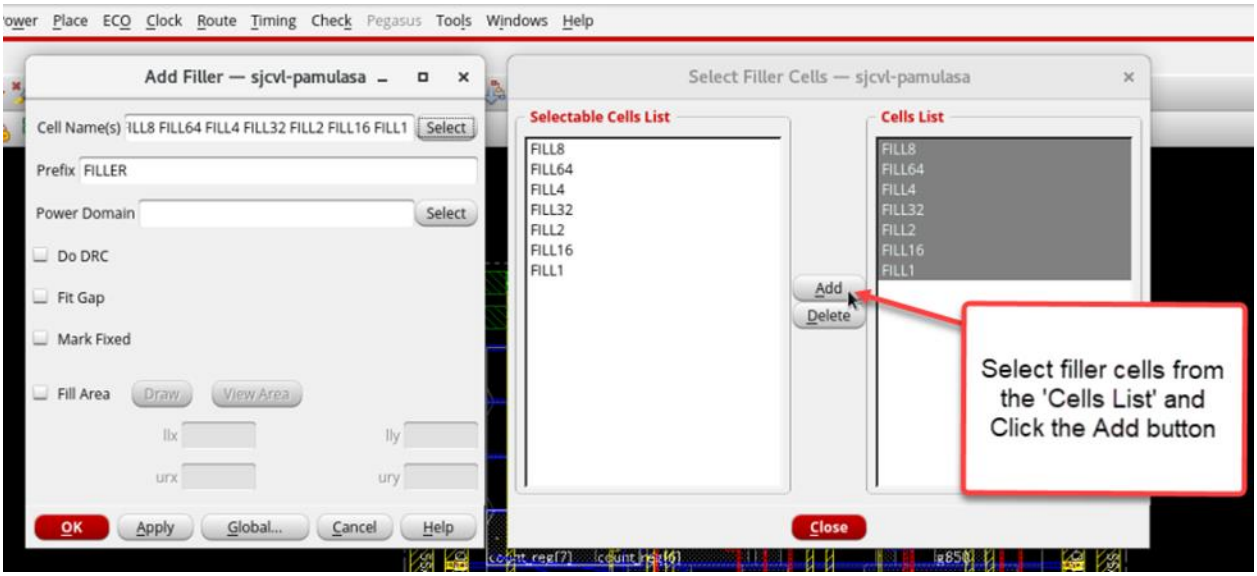
2. Click on the **Select** option in the Cell Name(s) field to choose the filler cells from the list.




This will bring up the **Select Filler Cells** window, as shown below.



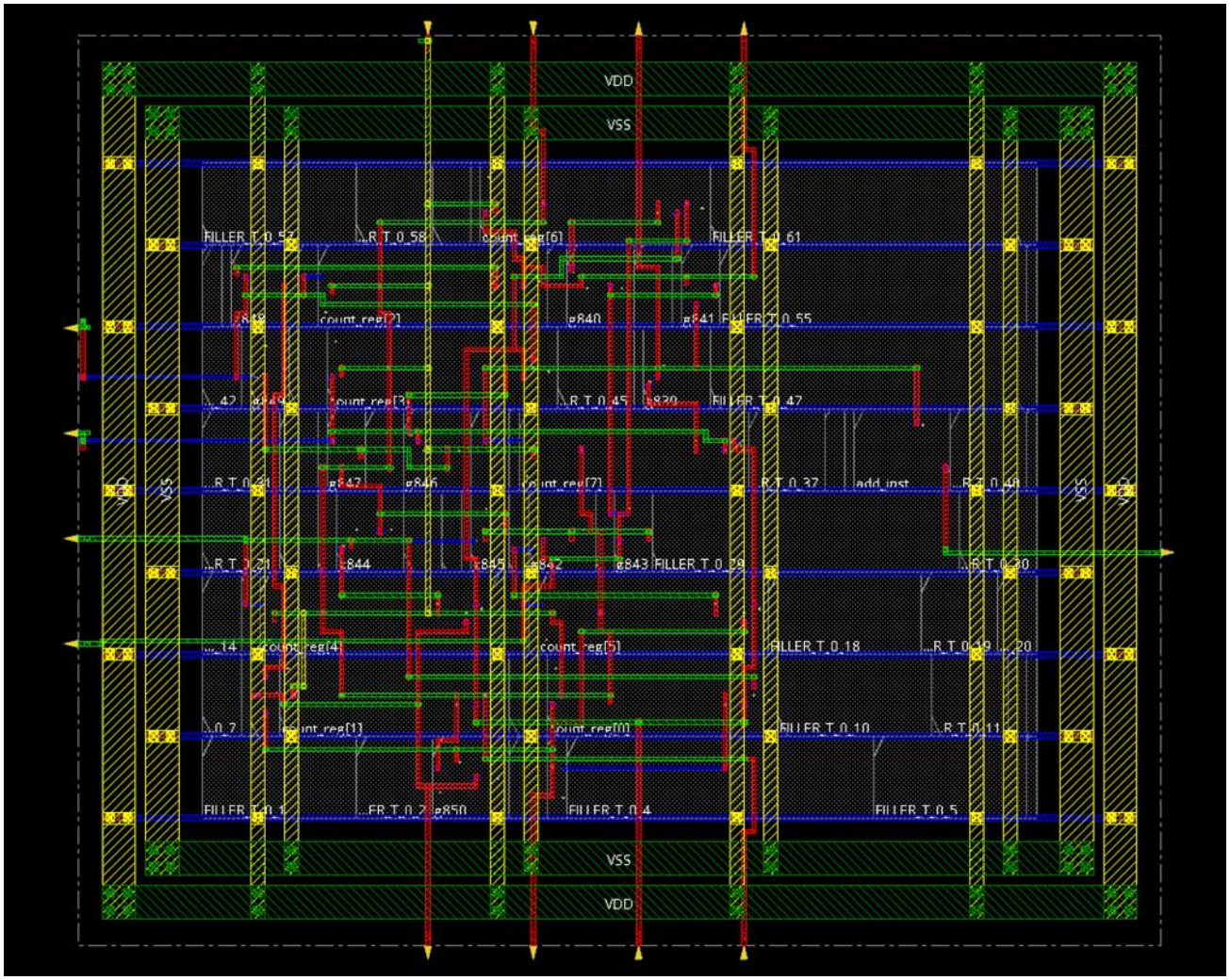
3. Add filler cells from the Cells list.




4. **Close** the **Select Filler Cells** window.

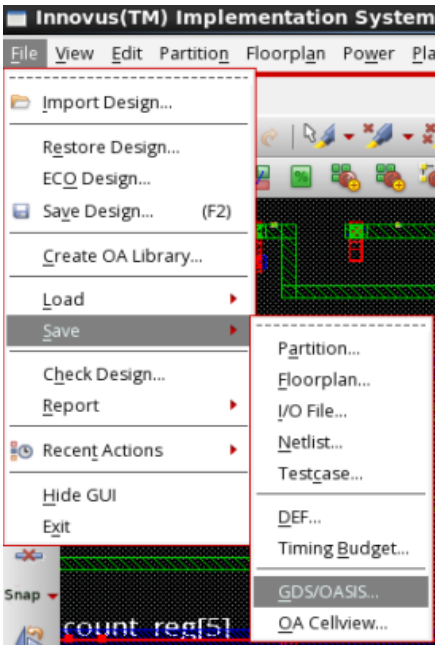
5. Add a prefix name.

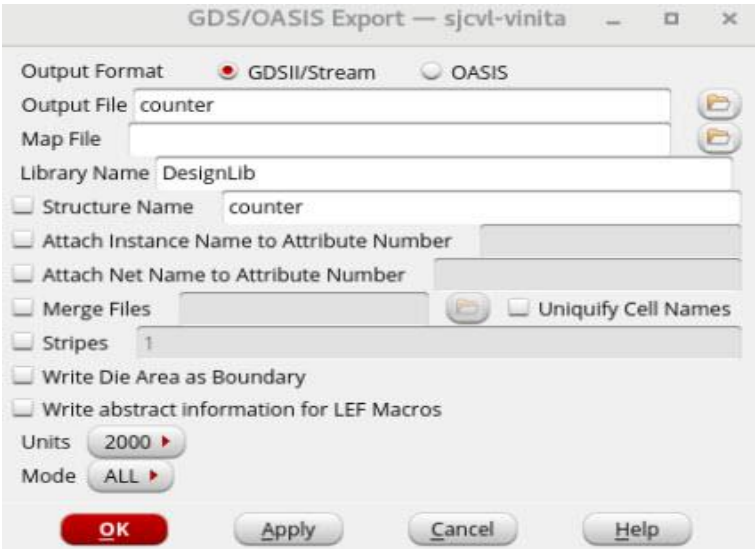
6. Click on **OK** to run the filler cell placement.



## Generating a Stream File

1. Select **File-Save-GDS/OASIS**.




   a. This will bring up the GDS/OASIS Export form.

   b. Enter the **counter** in the Output File field.

   c. Click **OK**.

2. Close the Innovus software.

End of Lab

# Module 9: Gate-Level Simulation

# Lab 9-1 Running Gate-Level Simulations on a Simple Counter Design

**Objective: To run gate-level simulations on a simple counter design using the Xcelium™ Simulator tool.**

---

This lab uses the following software:

- XCELIUM 23.09 (23.09.001)

## What Is Gate-Level Simulation (GLS)?

GLS is a step in the design flow to ensure that the design meets the functionality after synthesis or after placement and routing activities. We need a synthesized/post-routed netlist, a testbench, and an SDF (Standard Delay Format) file. The SDF will have all the delay information for the cell and the wire.

1. Here we will make use of the same testbench that we used for the functional simulation with some changes in the testbench. That is, we have to use the *$sdf_annotate* system task to call the *sdf* file inside the testbench.

2. We will perform gate-level simulation inside the *gate_level_simulation* directory under counter_database.

3. Files present inside the *gate_level_simulation* directory are listed below:
   - **Counter_netlist.v –** Netlist after synthesis with DFT (netlist after physical design also can be used).
   - **Counter_test.v –** Testbench with *$sdf_annotate* system task to input SDF file.
   - **slow_vdd1v0_basicCells.v** – Simulation library in *.v* format.
   - **delays.sdf** – SDF file generated during synthesis (SDF can also be generated after physical design).

## SDF Annotation

Modify the testbench to include the SDF configuration, as shown below. In *counter_test.v* available inside the *gate_level_simulation* directory, the SDF configuration system task is already included.

```
`timescale 1ns/10ps
module counter_test;
reg clk, rst;
wire [7:0] count;
wire scan_out;
reg scan_in, SE;

counter counter1(clk,rst,count,SE,scan_in,scan_out);

`ifdef SDF_TEST
initial
begin
$sdf_annotate("delays.sdf",counter_test.counter1,,"sdf.log","MAXIMUM");
end
`endif

initial
begin
clk=0;
rst=0;
#10 rst=1;
#1000 $stop;
end


always #5 clk=~clk;

endmodule
```

```
$sdf_annotate (“sdf_file”
{, module_instance}
{, “config_file”}
{, “log_file”}
{, “mtm_spec”}
{, “scale_factors”}
{, “scale_type”});
```

**Note:** We must specify the arguments to the *$sdf_annotate* system task in the order shown in the syntax. We can skip an argument specification, but the number of comma separators must maintain the argument sequence. For example, to specify only the first and last arguments, use the following syntax:

```
$sdf_annotate (“sdf_file”,,,,,, “scale_type”);
```

```
$sdf_annotate arguments:
```

- **“sdf_file”**: The full or relative path of the SDF file. This argument is required and must be in quotation marks. We can specify the file name with the *+sdf_file* plus option on the command line.

- **module_instance** (optional): Specifies the scope in which the annotation takes place. The names in the SDF file are relative paths to the module_instance with respect to the entire Verilog HDL description. The SDF Annotator uses the hierarchy level of the specified instance for running the annotation. Array indexes (module_instance [index]) are permitted in the scope. If we do not specify module_instance, the SDF Annotator uses the module containing the call to the *$sdf_annotate* system task as the module_instance for annotation.

- **"config_file"** (optional): The name of the configuration file, specified in quotation marks, that the SDF Annotator reads before annotating begins. If we do not specify config_file, the SDF Annotator uses the default settings.

- **"log_file"** (optional): The name of the log file specified in quotation marks that the SDF Annotator generates during annotation. Also, you must specify the *+sdf_verbose* plus option on the command line to generate a log file. If we do not specify a log file name but specify the *+sdf_verbose* plus option, the SDF Annotator creates a default log file called *sdf.log*.

- **"mtm_spec"** (optional): One of the following keywords, specified in quotation marks, indicates the delay values that are annotated to the Verilog family tool.

| Keyword | Description |
|---|---|
| `MAXIMUM` | Annotates the maximum delay value |
| `MINIMUM` | Annotates the minimum delay value |
| `TOOL_CONTROL (default)` | Annotates the delay value that is determined by the Verilog-XL and Verifault-XL command-line options (+mindelays, +typdelays or +maxdelays); minimum, typical, and maximum values are always annotated to Veritime. If none of the TOOL_CONTROL command-line options is specified, then the default keyword is TYPICAL. |
| `TYPICAL` | Annotates the typical delay value. |

- **"scale_factors"** (optional): The minimum, typical, and maximum timing data values, specified in quotation marks, are expressed as a set of three positive real number multipliers (min_mult:typ_mult:max_mult), e.g., 1.6:1.4:1.2. If we do not specify values, the default values are 1.0:1.0:1.0 for minimum, typical, and maximum values. The SDF Annotator uses these values to scale the minimum, typical, and maximum timing data from the SDF file before they are annotated to the Verilog family tool.

- **"scale_type"** (optional): One of the following keywords, specified in quotation marks, to scale the timing specifications in SDF, which are annotated to the Verilog family tool.

| Keyword | Description |
|---|---|
| FROM_MAXIMUM | Scales from the maximum timing specification. |
| FROM_MINIMUM | Scales from the minimum timing specification. |
| FROM_MTM (*default*) | Scales from the minimum, typical, and maximum timing specifications. This is the default. |
| FROM_TYPICAL | Scales from the typical timing specification. |

## Simulating the Netlist with the *xrun* Command

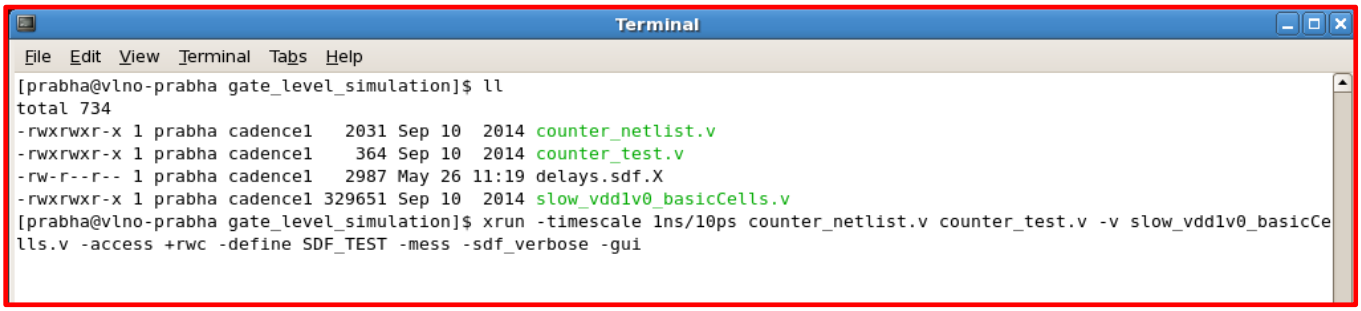
1. Change to the working directory:

```
cd gatfe_level_simulation
```

2. Execute the following *xrun* command:

```
xrun -timescale 1ns/10ps counter_netlist.v counter_test.v -v
   slow_vdd1v0_basicCells.v -access +rwc -define SDF_TEST -mess -gui
```

- **-timescale**: To mention the time unit and time precision.
- **-access**: Passed to the elaborator to provide read access to simulation objects.
- **-gui**: To invoke the *xrun* in *gui* mode.
- **-mess**: To display all the messages in detail.
- **-define**: To provide SDF definition present in the testbench.
- **-v**: To provide a library in ".v" format.



```
Elaborating the design hierarchy:
Top level design units:
        counter_test
Reading SDF file from location "delays.sdf"
Writing compiled SDF file to "delays.sdf.X".
Annotating SDF timing data:
        Compiled SDF file:     delays.sdf.X
        Log file:              sdf.log
        Backannotation scope:  counter_test.counter1
        Configuration file:
        MTM control:           MAXIMUM
        Scale factors:
        Scale type:
Annotation completed successfully...
SDF statistics: No. of Pathdelays = 57  Annotated = 100.00% -- No. of Tchecks = 96  Annotated = 66.67%
                              Total      Annotated      Percentage
         Path Delays             57             57          100.00
               $hold              8              0            0.00
            $removal              8              8          100.00
              $width             24              0            0.00
           $recovery              8              8          100.00
          $setuphold             48             48          100.00
Building instance overlay tables: ..................... Done
```

```
        Building instance specific data structures.
        Loading native compiled code:     ..................... Done
        Design hierarchy summary:
                                  Instances  Unique
                Modules:                 24       8
                UDPs:                    16       2
                Primitives:              79       5
                Timing outputs:          22       5
                Registers:               12       7
                Scalar wires:            43       -
                Always blocks:            1       1
                Initial blocks:           2       2
                Cont. assignments:        0       1
                Pseudo assignments:       4       4
                Timing checks:          144      19
                Interconnect:            67       -
                Delayed tcheck signals:  40      18
                Simulation timescale:    10ps
        Writing initial simulation snapshot: worklib.counter_test:v
-------------------------------------
Relinquished control to SimVision...
xcelium>
xcelium> source /grid/avs/install/xcelium/1704/latest/tools/xcelium/files/xmsimrc
xcelium>
```

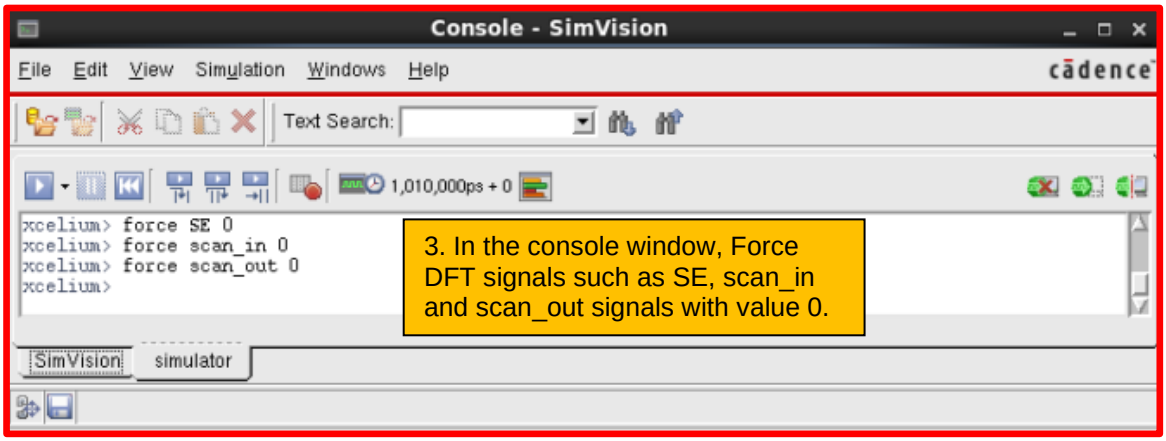
cadence

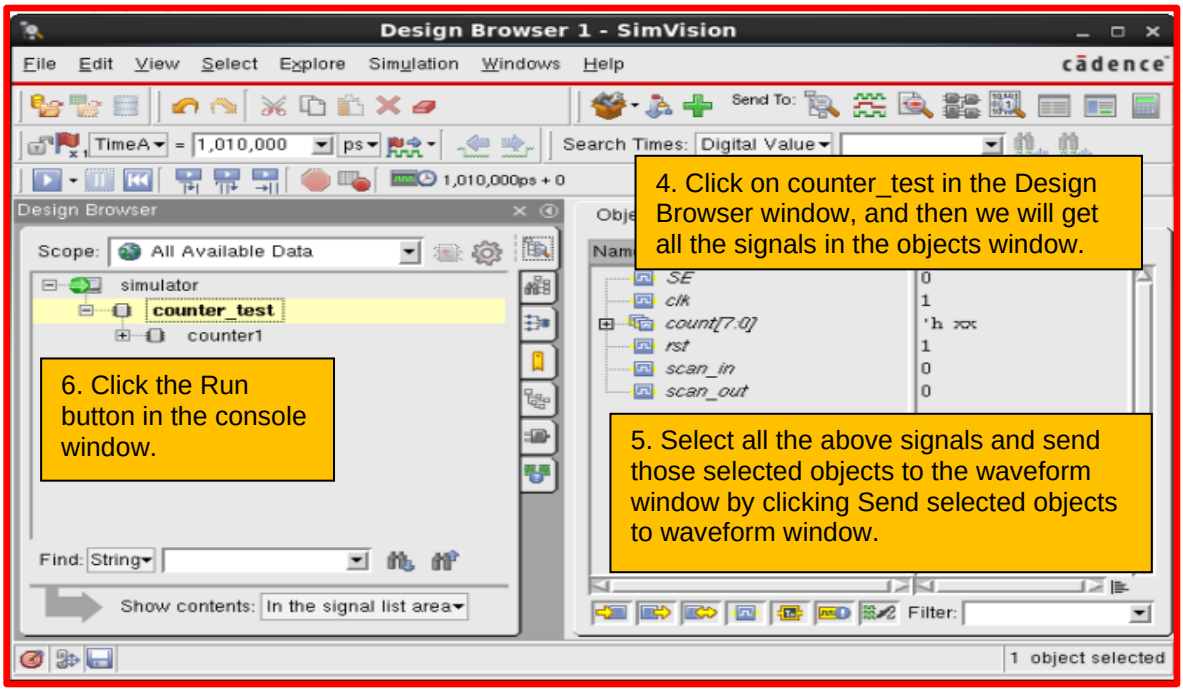
**Simulation Analysis Environment**
**SimVision 20.09-s003**

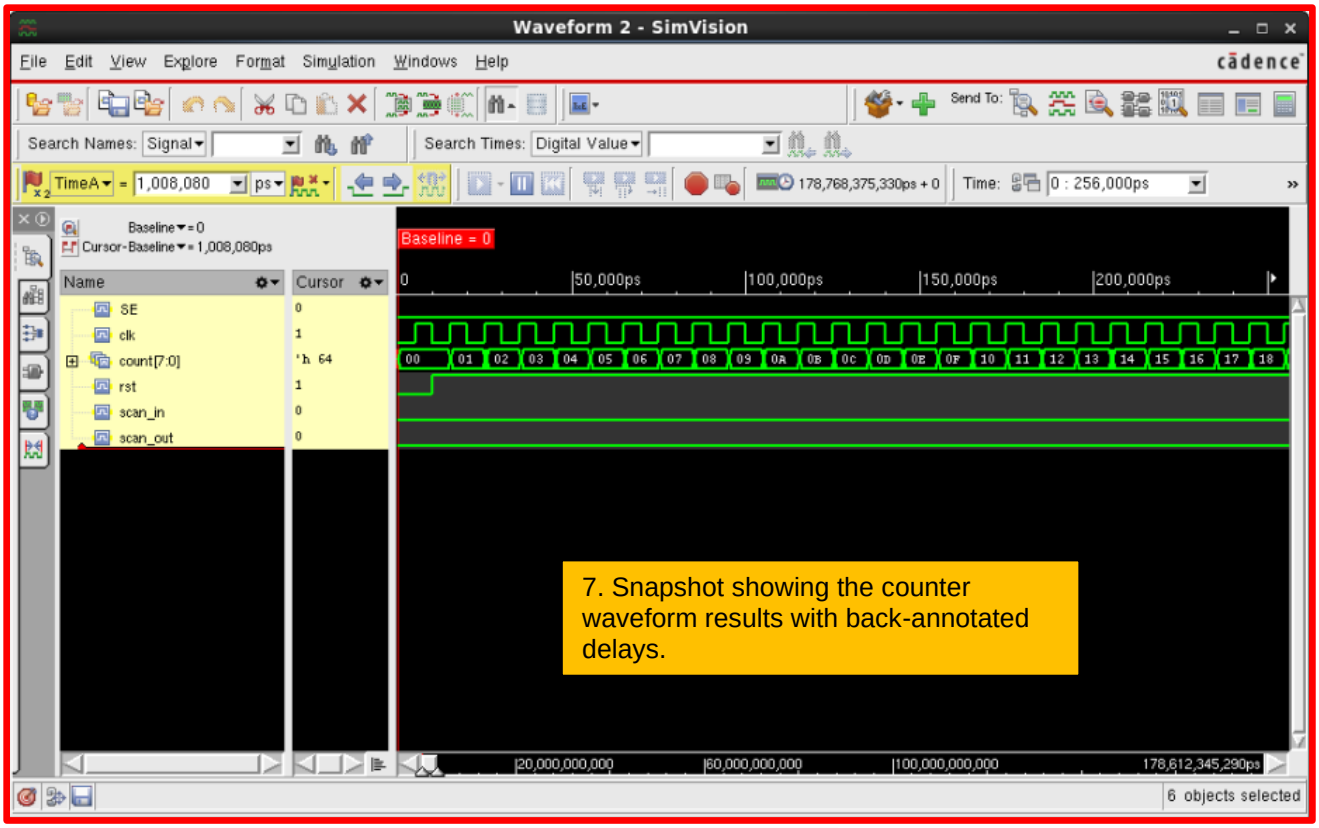
Loading Default layout...

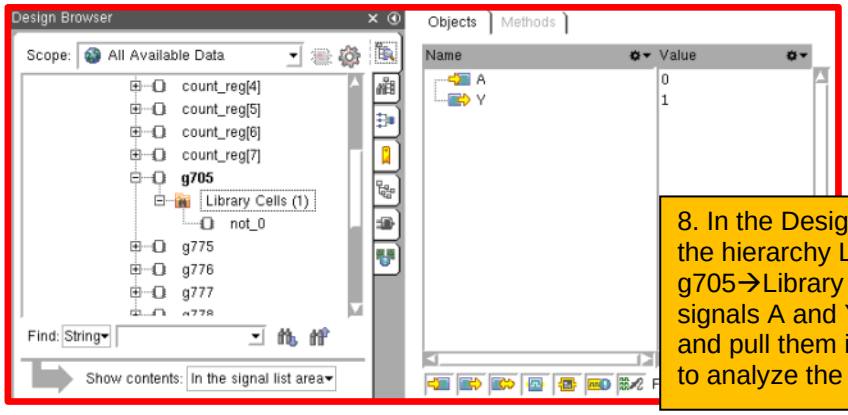


2. SimVision™ starts with the Console and Design Browser windows opening as seen below.






End of Lab

# Module 10: Timing Analysis and Debug

# Lab 10-1 Using Global Timing Debug Interface to Debug Timing Results

**Objective: To run the timing analysis and fix a couple of timing violations.**

---

Tempus™ Timing Signoff Solution is a timing signoff tool used to verify that the design meets your timing goals. In this lab, you will first rerun the previous session of Place & Route and continue it with running Tempus timing analysis inside of Innovus™.

This lab uses the following software:

- INNOVUS231 (23.10-p003_1)
- SSV231 (23.10-p001_1)

You will then run Tempus in the Timing Signoff mode.

- You can start the Tempus tool using the command *tempus -stylus*.
- You can start the Tempus TSO software using the command *tempus -stylus -eco*.

The following files are from the physical design lab and will serve as inputs to Tempus:

- **counter_netlist.v** – Gate-level netlist output after synthesis.
- **counter_sdc.sdc** – Constraint file generated during synthesis.
- **counter.view** – The implementation view definitions listed in this MMMC file.
- ***gsclib045_tech.lef, gsclib045_macro.lef*** – LEF files used in physical design.
- ***slow_vdd1v0_basicCells.lib*** and ***fast_vdd1v0_basicCells.lib*** – Timing libraries.

There are a few additional files in the directory that are not listed above.

## Running Timing Analysis and Debugging in the Innovus Session

Innovus is a cockpit for a lot of implementation and verification tools front-to-back.

In this section, you will be running a Tempus-style timing analysis inside of Innovus.

1. Let us rerun the previous session of Place & Route inside the *STA* directory instead of using a saved session:

```
LINUX# cd STA
LINUX# innovus -stylus -files runPnR.tcl
```

At the end of this session, the design is routed and timing analyzed.

A screenshot of runPnR.tcl is shown here.

```
set_db init_read_netlist_files ../physical_design/counter_netlist.v
set_db init_lef_files {../lef/gsclib045_tech.lef ../lef/gsclib045_macro.lef}
set_db init_power_nets VDD
set_db init_ground_nets VSS
set_db init_mmmc_files  counter.view
read_mmmc counter.view
read_physical -lef {../lef/gsclib045_tech.lef ../lef/gsclib045_macro.lef}
read_netlist ../physical_design/counter_netlist.v -top counter
init_design


connect_global_net VDD -type pg_pin -pin_base_name VDD -inst_base_name *
connect_global_net VSS -type pg_pin -pin_base_name VSS -inst_base_name *

#Create Floorplan
create_floorplan -core_margins_by die -site CoreSite -core_density_size 1 0.7 2.5 2.5 2.5 2.5

#Pin Assignment
read_io_file pins.io

#Power Planning
set_db add_rings_skip_shared_inner_ring none ; set_db add_rings_avoid_short 1 ; set_db add_rings_ignore_rows 0 ; set_db add_rings_extend_over_row 0

#Add Rings
add_rings -type core_rings -jog_distance 0.6 -threshold 0.6 -nets {VDD VSS} -follow core -layer {bottom Metal11 top Metal11 right Metal10 left Metal10} -width 0.7 -spacing
.4 -offset 0.6

#Add Stripes
add_stripes -block_ring_top_layer_limit Metal11 -max_same_layer_jog_length 0.44 -pad_core_ring_bottom_layer_limit Metal9 -set_to_set_distance 5 -pad_core_ring_top_layer_lim
it Metal11 -spacing 0.4 -merge_stripes_value 0.6 -layer Metal10 -block_ring_bottom_layer_limit Metal9 -width 0.3 -nets {VDD VSS}

# Create Power Rails with Special Route
route_special -connect core_pin -layer_change_range { Metal1(1) Metal11(11) } -block_pin_target nearest_target -core_pin_target first_after_row_end -allow_jogging 1 -crosso
ver_via_layer_range { Metal1(1) Metal11(11) } -nets { VDD VSS } -allow_layer_change 1 -target_via_layer_range { Metal1(1) Metal11(11) }
# Read the Scan DEF
```

2. Set the timing analysis mode by running the following commands:

   ```
   set_db timing_analysis_type ocv
   ```

3. Run setup-and-hold timing analysis by running the following commands:

   ```
   time_design -post_route
   ```

   *Are there any setup violations?*

   Answer: _____

   As you can see in the following timing summary, the design doesn't have any violating paths in setup mode.

```
------------------------------------------------------------------
         time_design Summary
------------------------------------------------------------------

Setup views included:
 wc

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  8.395  |  8.395  |  8.726  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   40    |   15    |   25    |
+--------------------+---------+---------+---------+
```

   Let's run the hold check using the command:

   ```
   time_design -post_route -hold
   ```

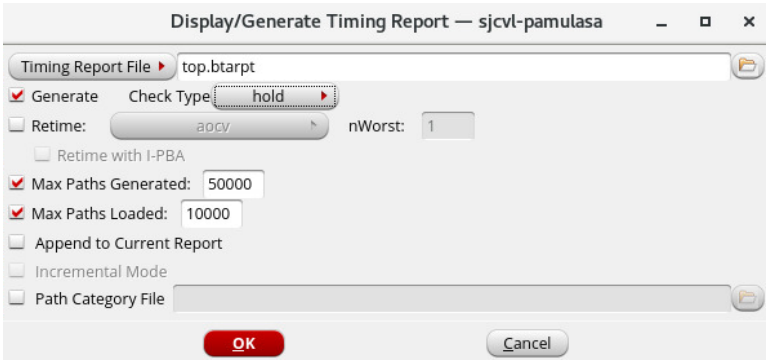
```
------------------------------------------------------------------
         time_design Summary
------------------------------------------------------------------

Hold views included:
 bc

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):| -0.093  | -0.024  | -0.093  |
|           TNS (ns):| -0.983  | -0.156  | -0.826  |
|    Violating Paths:|   17    |    8    |    9    |
|          All Paths:|   24    |   15    |    9    |
+--------------------+---------+---------+---------+
```

**Note:** There are a lot of **hold violations** in the design. Let us see how to fix all of them in Tempus.

4. If the graphical interface is not open, start it using the following command:

   ```
   gui_show
   ```

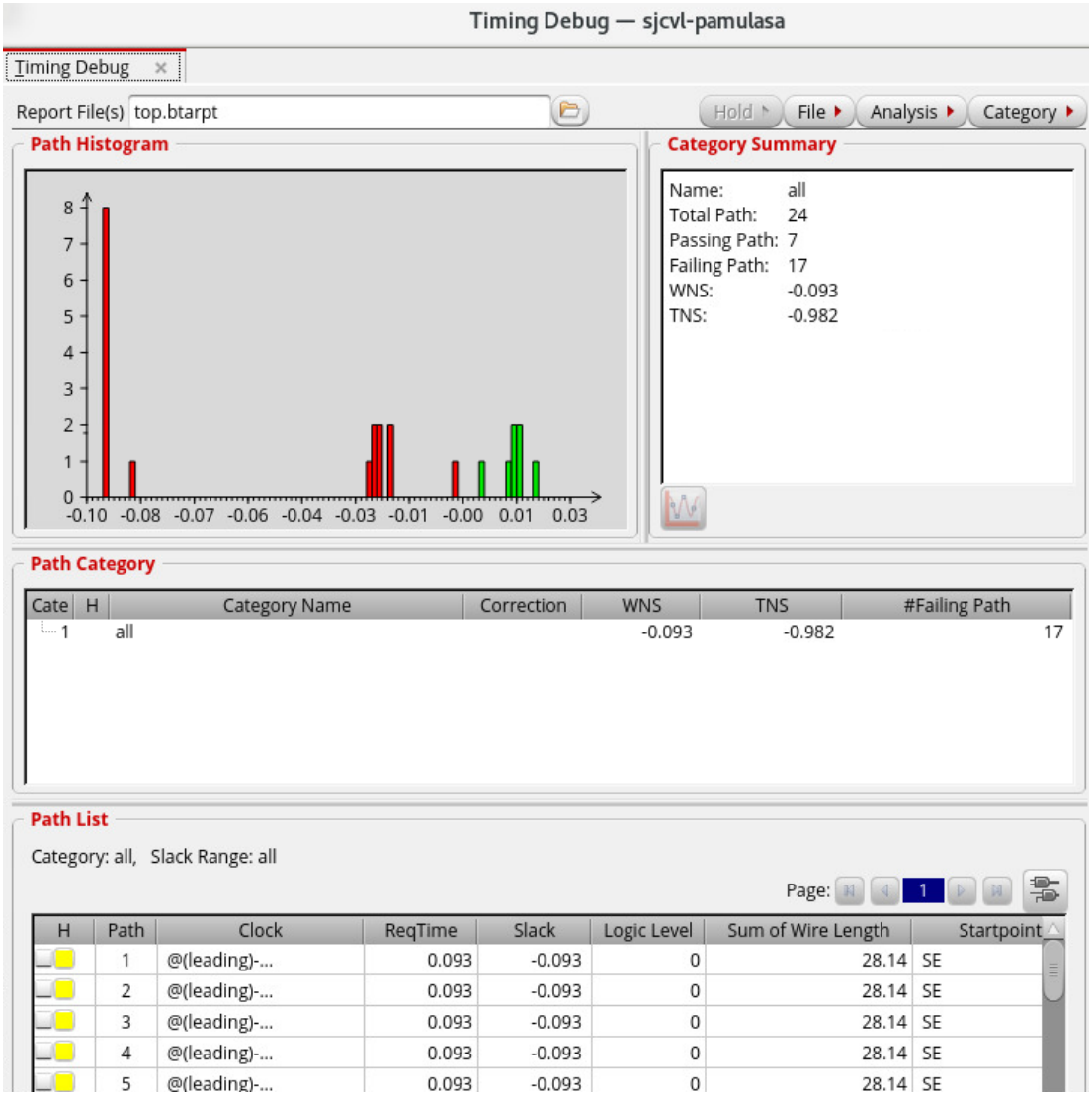
5. Run timing analysis using the **Timing** – **Debug Timing** menu.




   a. Select **hold** in the Check Type field.

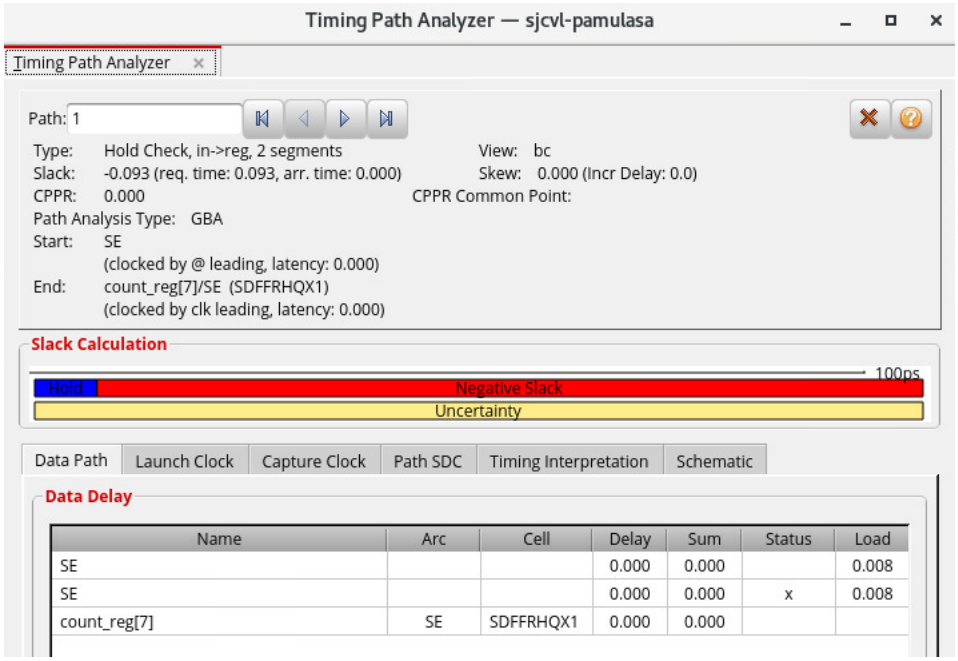
   b. Click **OK**.

The Timing Debug window opens.




c. **Right**-click on path **#1** in the Path List and select **Show Timing Path Analyzer**.

You will see the Timing Path Analyzer window.




**Note:** From the Timing Path Analyzer, if you want to debug violating paths, you can **right**-click on any of the signals, and you will get additional options like interactive ECO, etc.

6. Save the design including the SPEF, DEF, and the libraries, by entering:

   ```
   write_db -rc_extract -def postRoute
   ```

7. Keep this session open if you would like for debugging purposes. You can close it later.

## Running an Independent Timing Analysis in Tempus

1. Start Tempus independently:

   ```
   tempus -stylus
   ```

2. Load the Innovus database into Tempus using the following command:

   ```
   read_db -physical_data postRoute
   ```

   This loads the entire design along with the physical layout.

   Open the **Layout** tab (click the  sign to see other available tabs) to confirm that the same layout from Innovus is also shown here in Tempus.

   Once the design is loaded successfully, then generate the following reports.

3. Check the analysis coverage using the following command:

   ```
   report_analysis_coverage
   ```

4. View the list of all constraint violations using the following command:

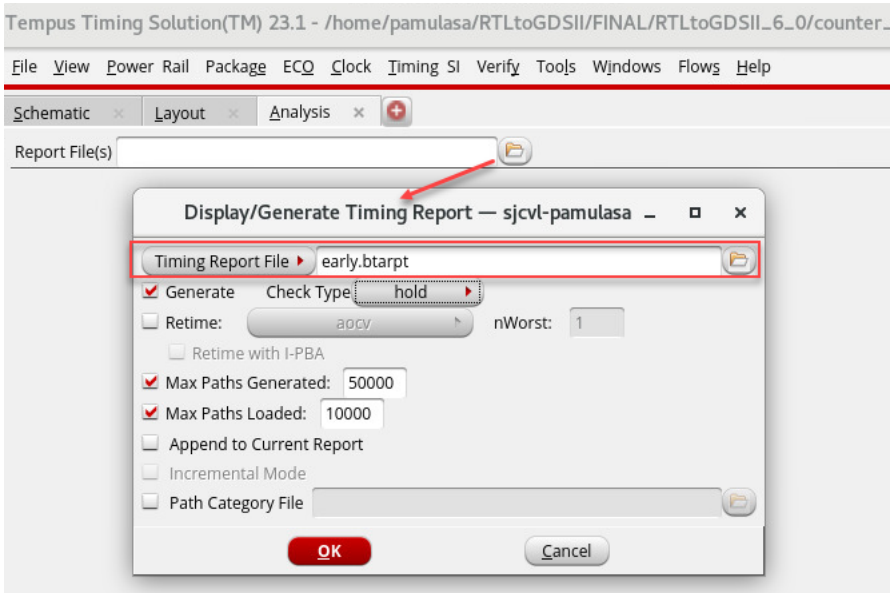
   ```
   report_constraint -all_violators
   ```

5. Report the worst slack time for setup and hold, respectively, using the following commands:

   ```
   report_timing -late
   report_timing -early
   ```

6. Open the **Analysis** tab (click the  sign and select **Analysis**).

   The Display/Generate dialog box appears.

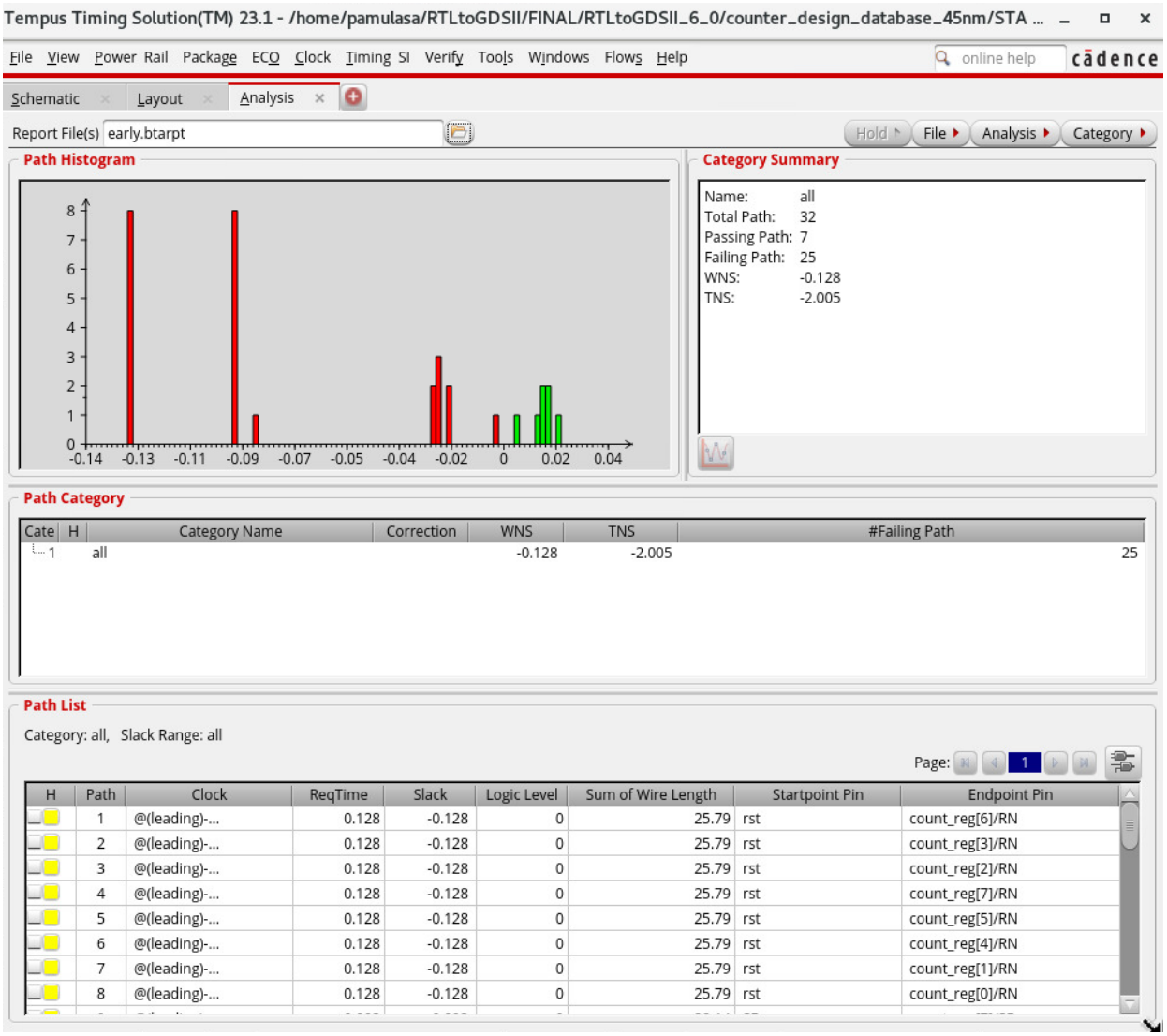



   a. Enter the Timing report file name **early.btarpt.**

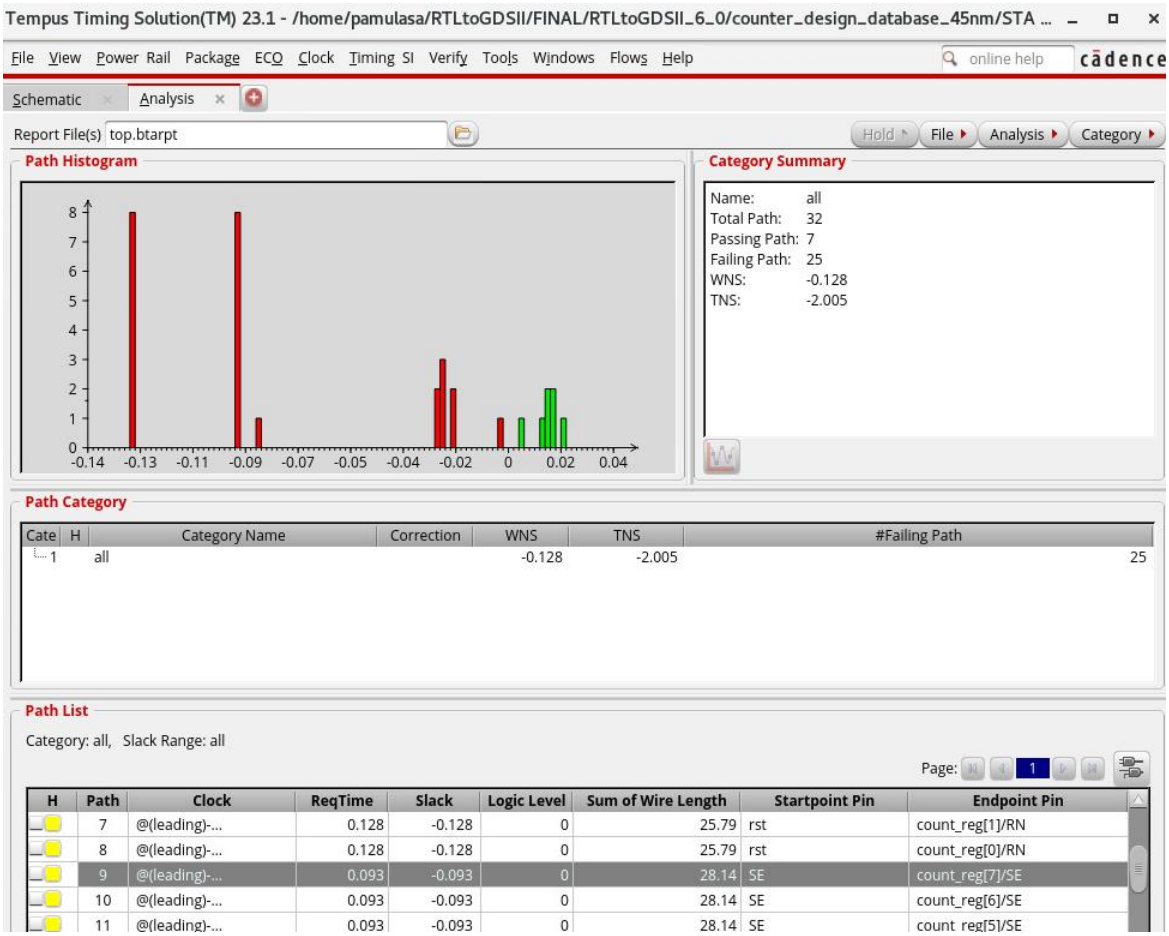
   b. Change Check Type **Hold.**

   c. Click **OK.**

After the timing analysis is done, you can see the histogram for the hold analysis.

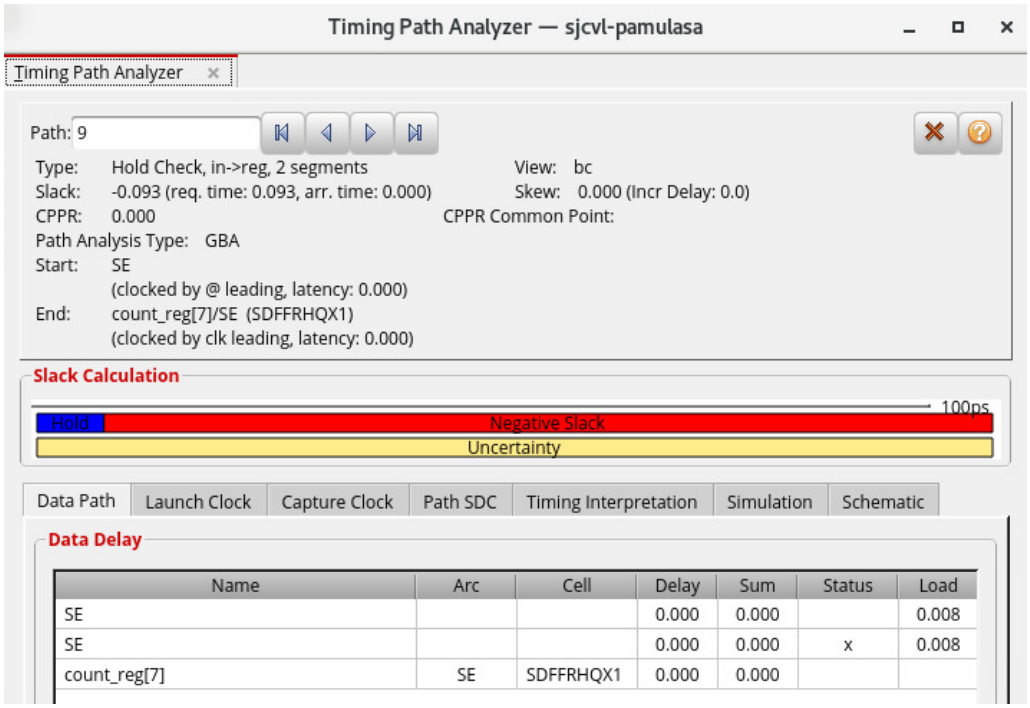


d. Analyze and fix the timing violation for the pin SE first (**path #9**).

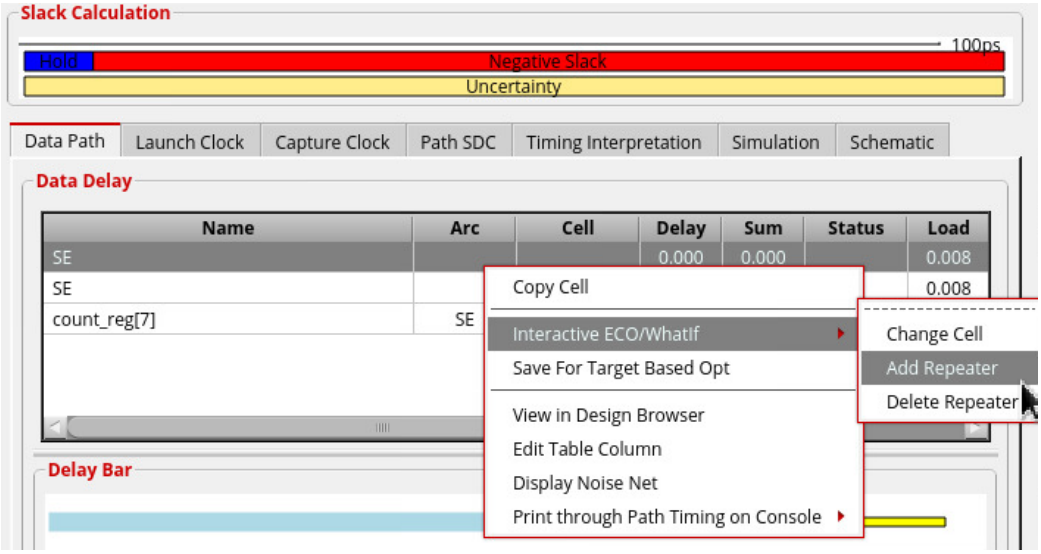


e. Browse through the Path List with Startpoint Pin as **SE (**path **#9)**, **right**-click on the path, and select **Show Timing Path Analyzer**.

You will see the Timing Path Analyzer window.



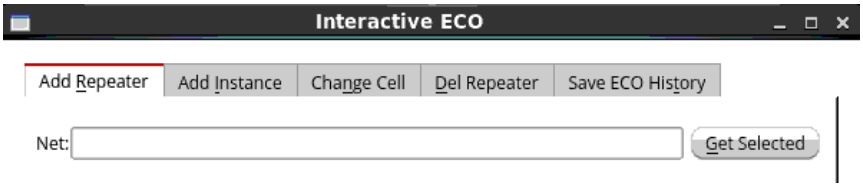

7. In the Timing Path Analyzer window, select the **SE** pin, **right**-click, and select **Interactive ECO/WhatIf** – **Add Repeater**.



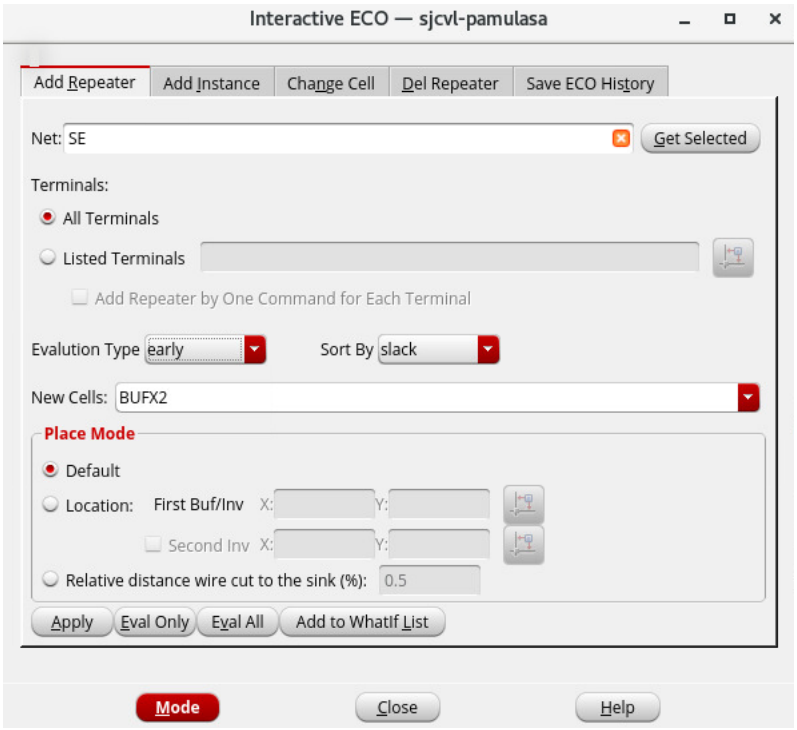

The Interactive ECO window opens, which will allow you to ***add repeaters***.

8. In the Interactive ECO window:

   a. Click **Get Selected** to populate the net information, or just enter **SE**.




   b. In the New Cell field, select **BUFX2**.




   c. Choose **early** from the *Evaluation Type*.

   d. Click **Eval All** to choose which buffer is suitable for the path that adds more delay and reduces the hold slack.

Notice the list of buffer suggestions appears at the Tempus terminal.

```
------------
@tempus 12> eco_add_repeater -cells BUFX2 -net SE -evaluate_all -sort_by slack -early
################################################################################
# Design Name: counter
# Design Mode: 65nm
# Analysis Mode: MMMC OCV
# Parasitics Mode: SPEF/RCDB
# Signoff Settings: SI On
################################################################################
-------------------------------------------------------------------------------------------------------------------
------------
Library-cell                            Area     Timing-slack   Worst-Slack-View            Worst-Slack-Dir
-------------------------------------------------------------------------------------------------------------------
------------
Reference                               0.0000     -0.0926      bc                          fall
CLKBUFX20(buf)                          8.2080     -0.0667      bc                          fall
BUFX20(buf)                             8.2080     -0.0666      bc                          fall
CLKBUFX16(buf)                          6.8400     -0.0666      bc                          fall
BUFX16(buf)                             6.8400     -0.0664      bc                          fall
CLKBUFX12(buf)                          5.1300     -0.0660      bc                          fall
CLKBUFX6(buf)                           3.0780     -0.0660      bc                          fall
BUFX12(buf)                             5.1300     -0.0659      bc                          fall
BUFX6(buf)                              3.0780     -0.0657      bc                          fall
BUFX8(buf)                              3.7620     -0.0643      bc                          fall
CLKBUFX8(buf)                           3.7620     -0.0643      bc                          fall
CLKBUFX4(buf)                           2.3940     -0.0594      bc                          fall
CLKBUFX3(buf)                           2.0520     -0.0592      bc                          fall
BUFX3(buf)                              2.0520     -0.0591      bc                          fall
BUFX4(buf)                              2.3940     -0.0591      bc                          fall
CLKBUFX2(buf)                           1.7100     -0.0520      bc                          fall
BUFX2(buf)                              1.7100     -0.0516      bc                          fall
-------------------------------------------------------------------------------------------------------------------
```

From the evaluation list, we can observe that Slack can be reduced to a maximum extent to **-0.0516** with the cell **BUFX2.**

**Note:** Add only BUFX* cells to fix the timing violation; do not add any CLKBUFX* cells to the design.

e. Now, select the **BUFX2** cell in the Interactive ECO.

f. Click **Eval Only**.

g. View the messages by the <**tempus**> prompt.

**Note:** Ensure that there is a change in the Slack (it is negative). The result should be an improvement in Slack **-0.0517** when you Apply the repeater addition.

```
@tempus 13> eco_add_repeater -cells BUFX2 -net SE -evaluate_only -sort_by slack -early
################################################################################
# Design Name: counter
# Design Mode: 65nm
# Analysis Mode: MMMC OCV
# Parasitics Mode: SPEF/RCDB
# Signoff Settings: SI On
################################################################################
-------------------------------------------------------------------------------------------------------------------
------------
Library-cell                            Area     Timing-slack   Worst-Slack-View            W
-------------------------------------------------------------------------------------------------------------------
------------
Reference                               0.0000     -0.0926      bc
BUFX2(buf)                              1.7100     -0.0517      bc
-------------------------------------------------------------------------------------------------------------------
------------
```

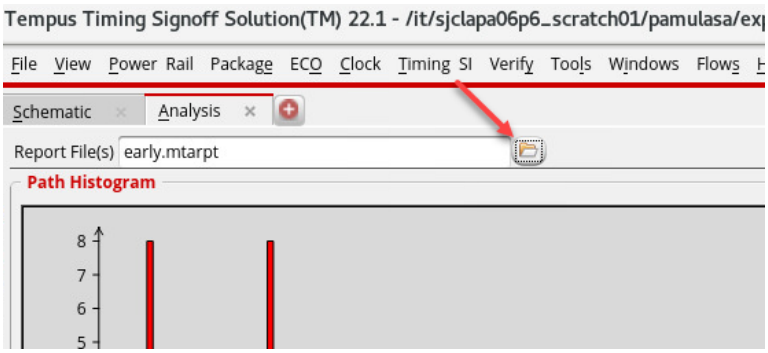
9. Click **Apply**.

   The slack improvement is clearly observable. Let us now analyze the results using the Timing Path Analyzer window.

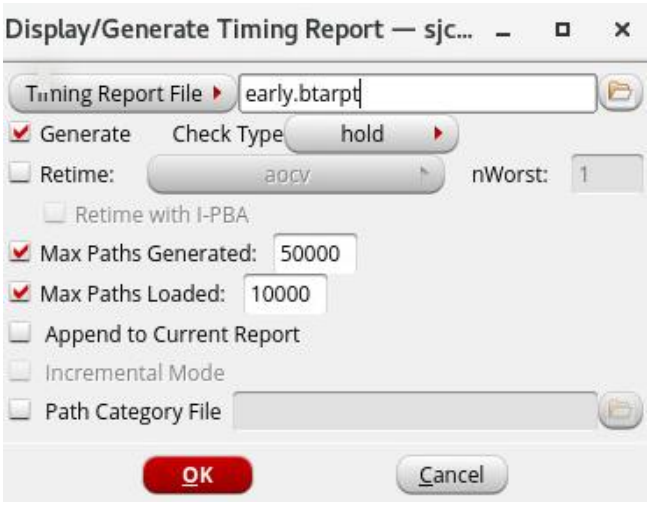
   **Note**: Ignore the *ERROR: (IMPSP-365)* if any.

10. Rerun the timing report using the below icon from the **Analysis** window and compare the WNS.

    a. Click the load icon from the *Report Files(s) field.*




   The Display/Generate Timing Report window appears.



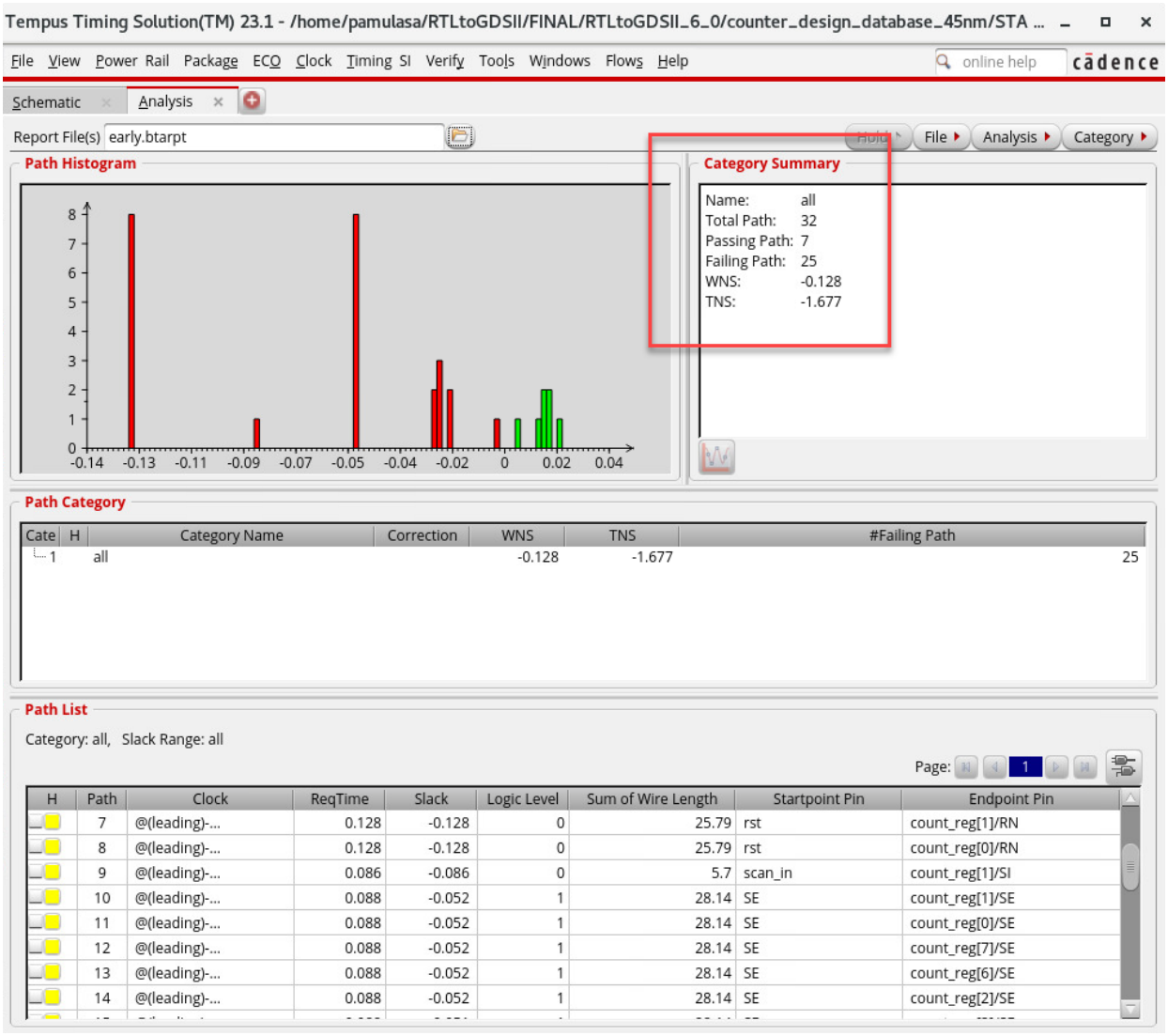

    b. Choose the Timing Report File ***early.btarpt.***

    c. Select **hold** from Check Type.

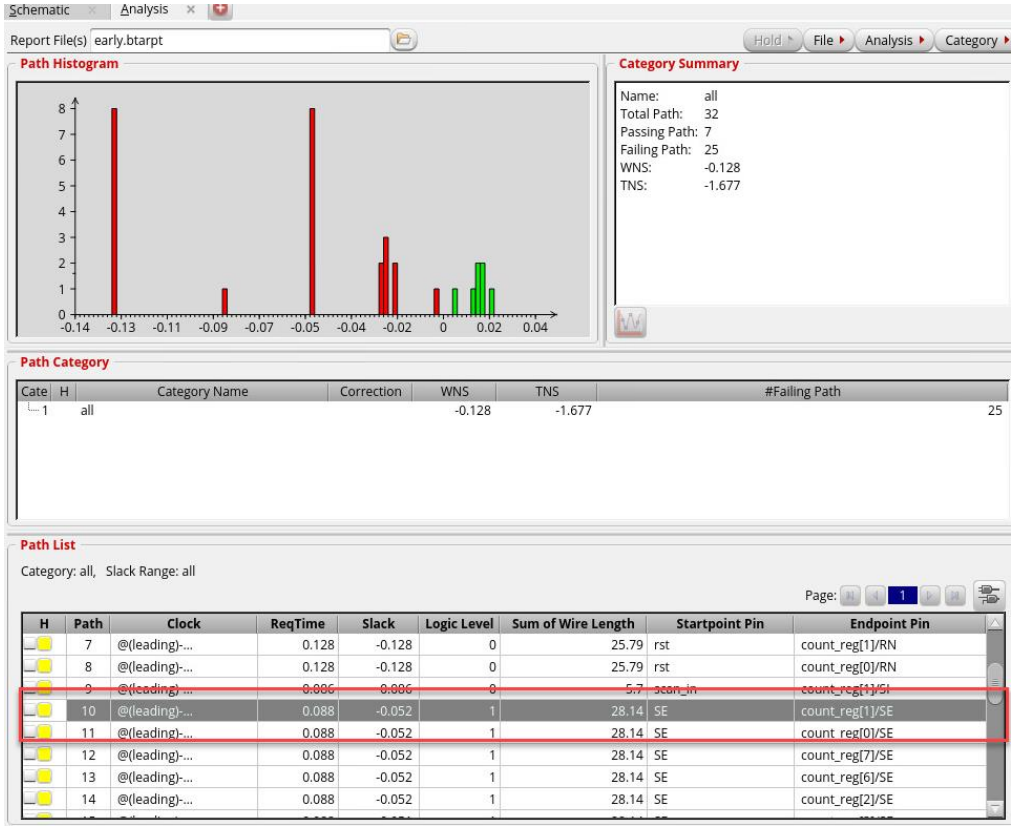
    d. Click **OK.**

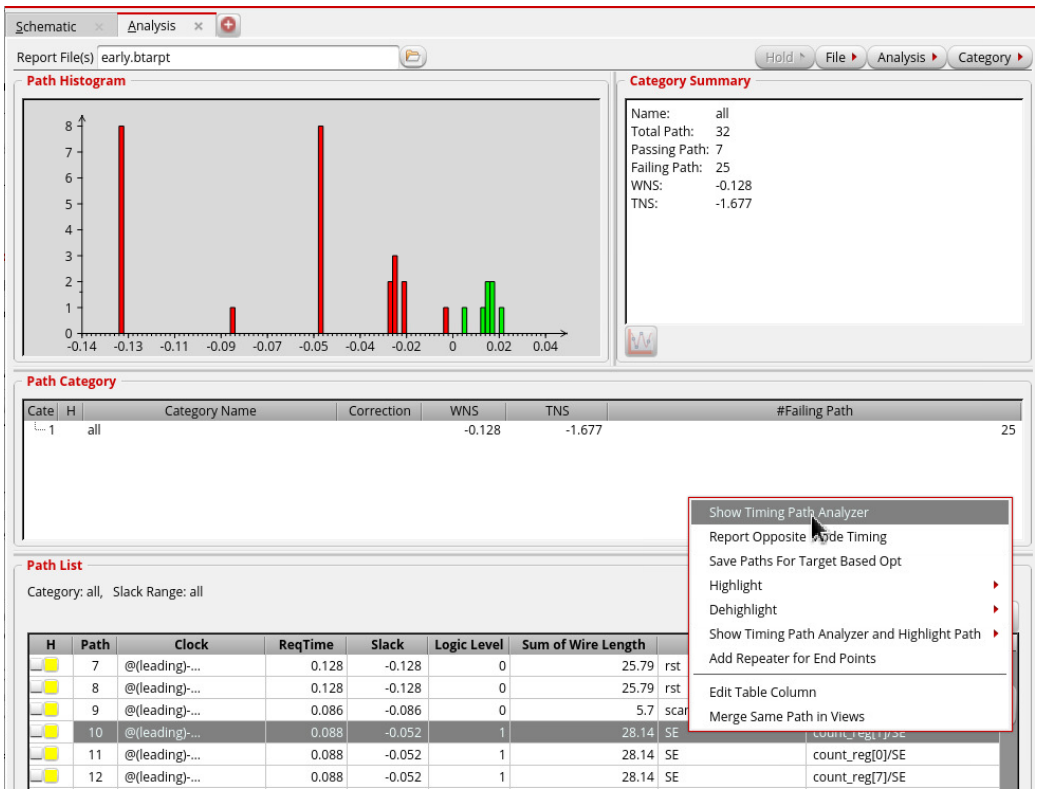
11. Notice the improvement in timing using the following screenshots.

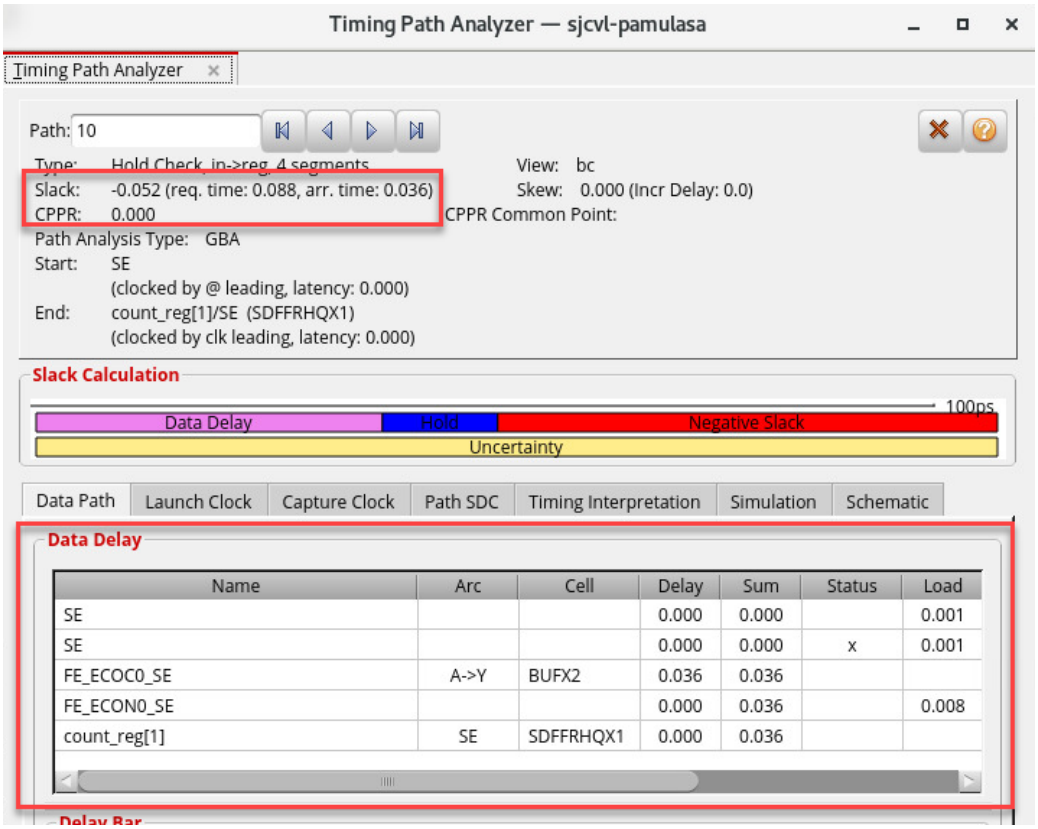
    Notice the improvement in the **WNS** in the **Analysis** window.



Notice the improvement in Hold Slack for the **SE** pin (path #**10**) in the **Timinig Path Analyzer** window.

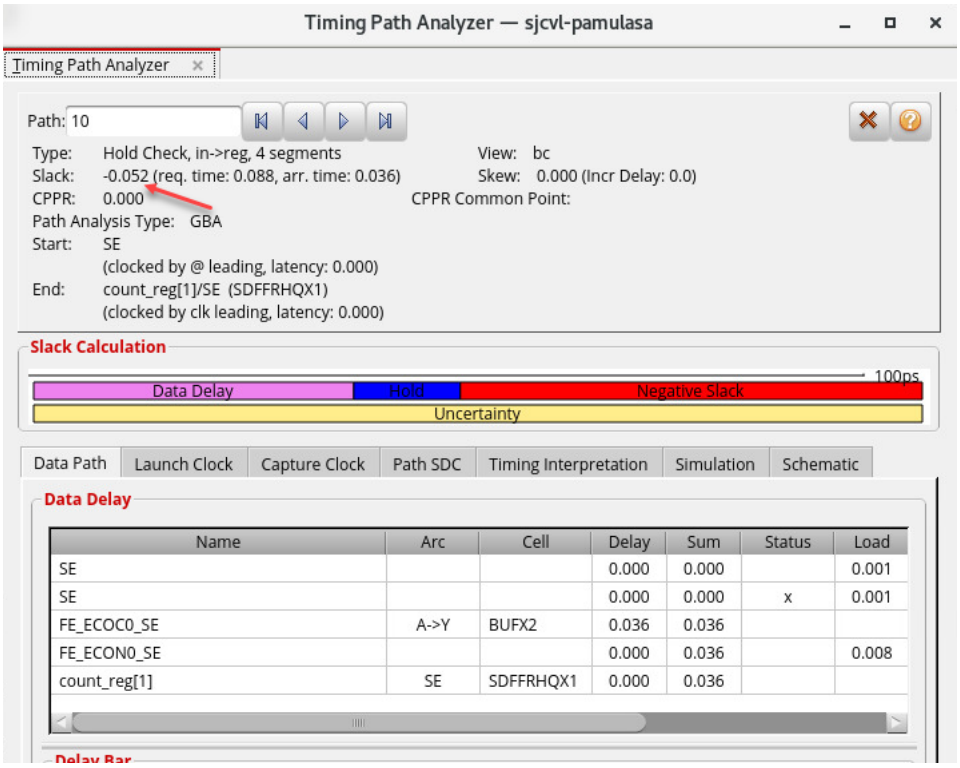





12. As we can see, the Slack is still negative (**-0.052**), and we need to add a buffer on the same net of the pin **SE** until we get the Zero/Positive Slack for the pin **SE.**

13. In the **Analysis** window, select the **SE** pin, **right-click** again on the path, and select **Show Timing Path Analyzer**.




14. Repeat Steps 8, 9, and 10 again by selecting the path of the **SE** pin and choosing **Add Repeater** from the Timing Path Analyzer window.

    Click **Eval all** and choose the buffer cell (BUFX*).

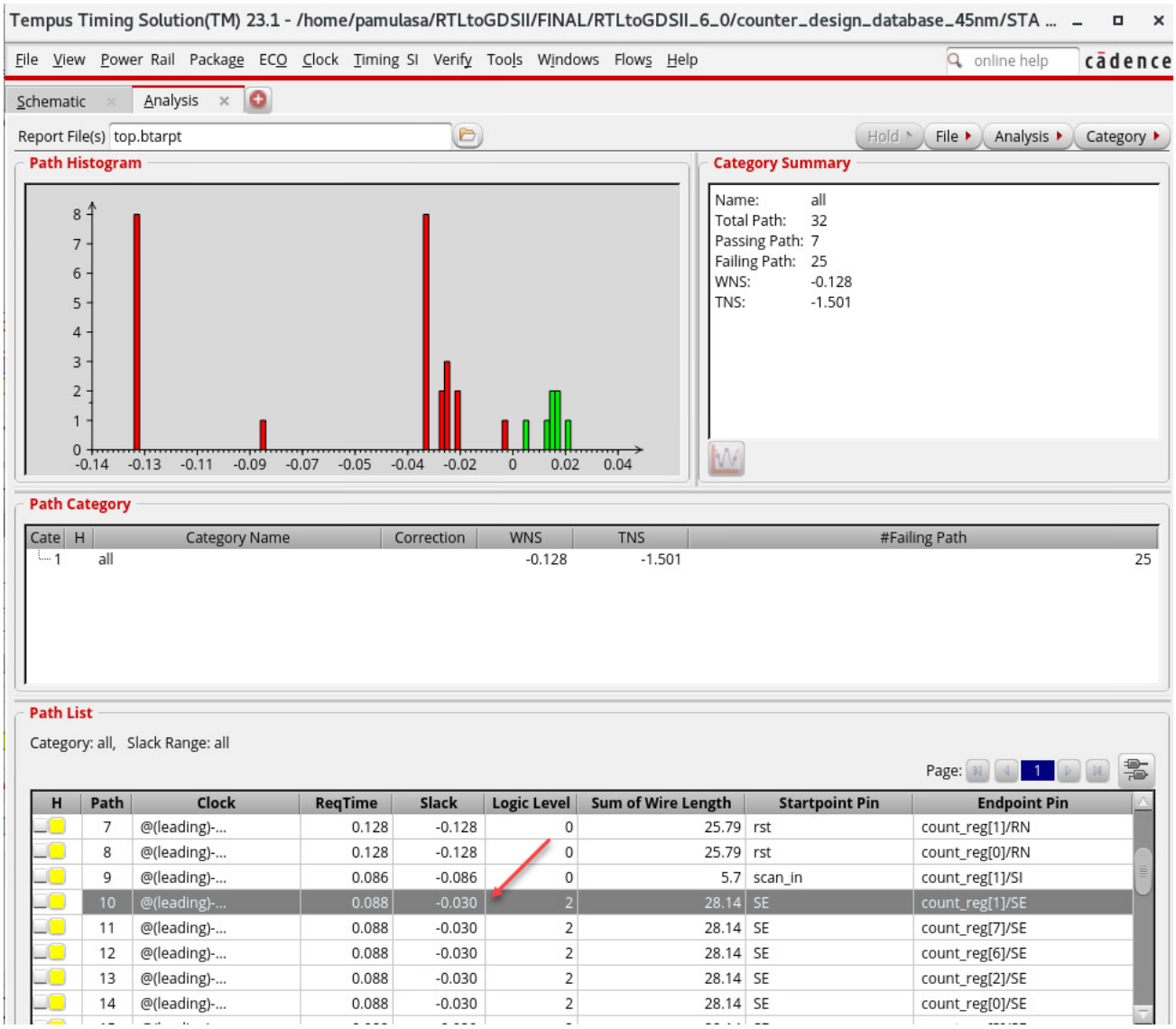
```
--------------------------------------------------------------------------------
@tempus 17> eco_add_repeater -cells BUFX2 -net SE -evaluate_all -sort_by slack -early
################################################################################
# Design Name: counter
# Design Mode: 65nm
# Analysis Mode: MMMC OCV
# Parasitics Mode: SPEF/RCDB
# Signoff Settings: SI On
################################################################################
--------------------------------------------------------------------------------
------------
Library-cell                              Area     Timing-slack   Worst-Slack-View
--------------------------------------------------------------------------------
------------
Reference                                 0.0000     -0.0516      bc
CLKBUFX6(buf)                             3.0780     -0.0320      bc
BUFX6(buf)                                3.0780     -0.0319      bc
BUFX3(buf)                                2.0520     -0.0313      bc
CLKBUFX3(buf)                             2.0520     -0.0313      bc
CLKBUFX2(buf)                             1.7100     -0.0298      bc
BUFX2(buf)                                1.7100     -0.0296      bc
CLKBUFX12(buf)                            5.1300     -0.0293      bc
CLKBUFX16(buf)                            6.8400     -0.0293      bc
BUFX12(buf)                               5.1300     -0.0292      bc
BUFX16(buf)                               6.8400     -0.0292      bc
BUFX8(buf)                                3.7620     -0.0291      bc
CLKBUFX20(buf)                            8.2080     -0.0291      bc
CLKBUFX8(buf)                             3.7620     -0.0291      bc
BUFX20(buf)                               8.2080     -0.0290      bc
CLKBUFX4(buf)                             2.3940     -0.0286      bc
BUFX4(buf)                                2.3940     -0.0285      bc
--------------------------------------------------------------------------------
```

Observe the **BUFX2** cell is the best delay cell to add to the path (**SE** pin) to reduce the Hold Timing Slack to **-0.0285.**

**Note:** If you have an area constraint or less area in your design, then choose a cell that adds moderate delay and less area according to the design requirement.

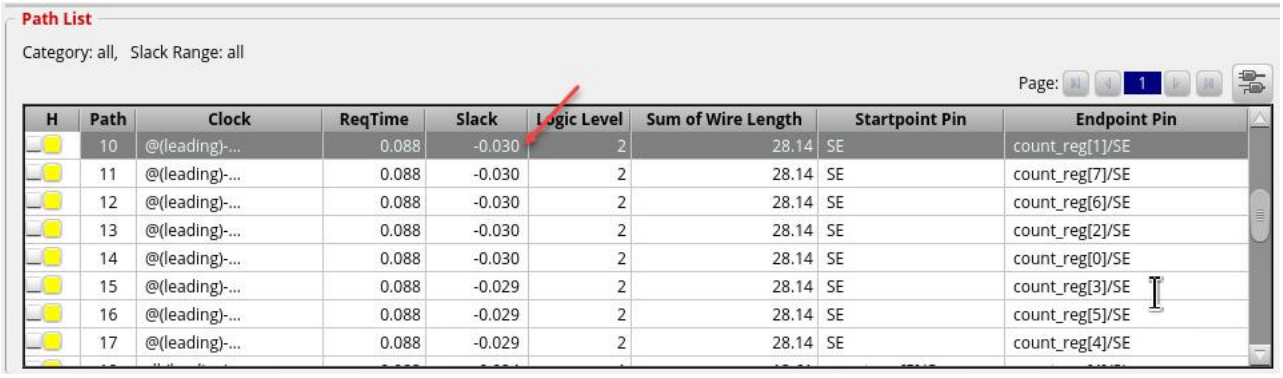
15. Click **Apply** to add a buffer.

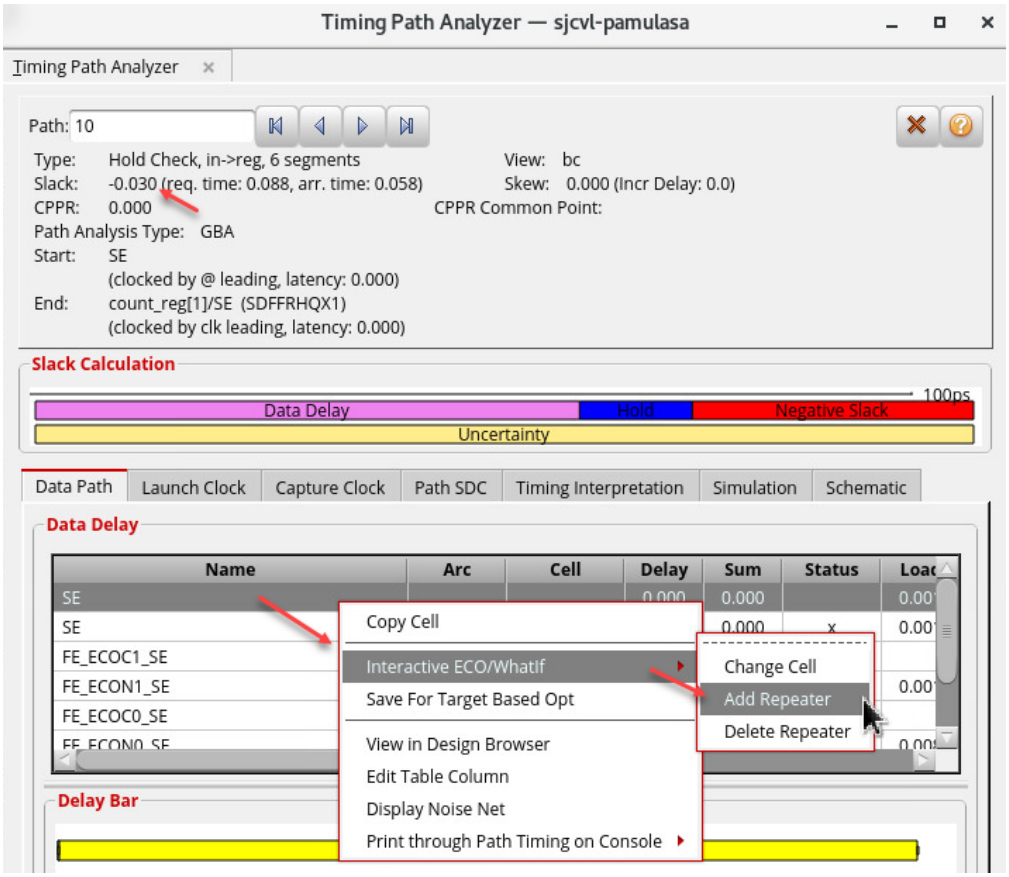
16. After adding the **BUFX2** cell, rerun the **Timing Debug** report as shown in **Step 10.**




**Note:** As we can see, the Slack is still negative (**-0.030**), and we need to add buffers on the same net of the pin **SE** until we get the Zero/Positive Slack for the pin **SE.**

17. Repeat Steps 7, 8, 9 and 10 again by selecting the path of the **SE** pin from the **Analysis** window and choosing **Add Repeater** from the Timing Path Analyzer window.



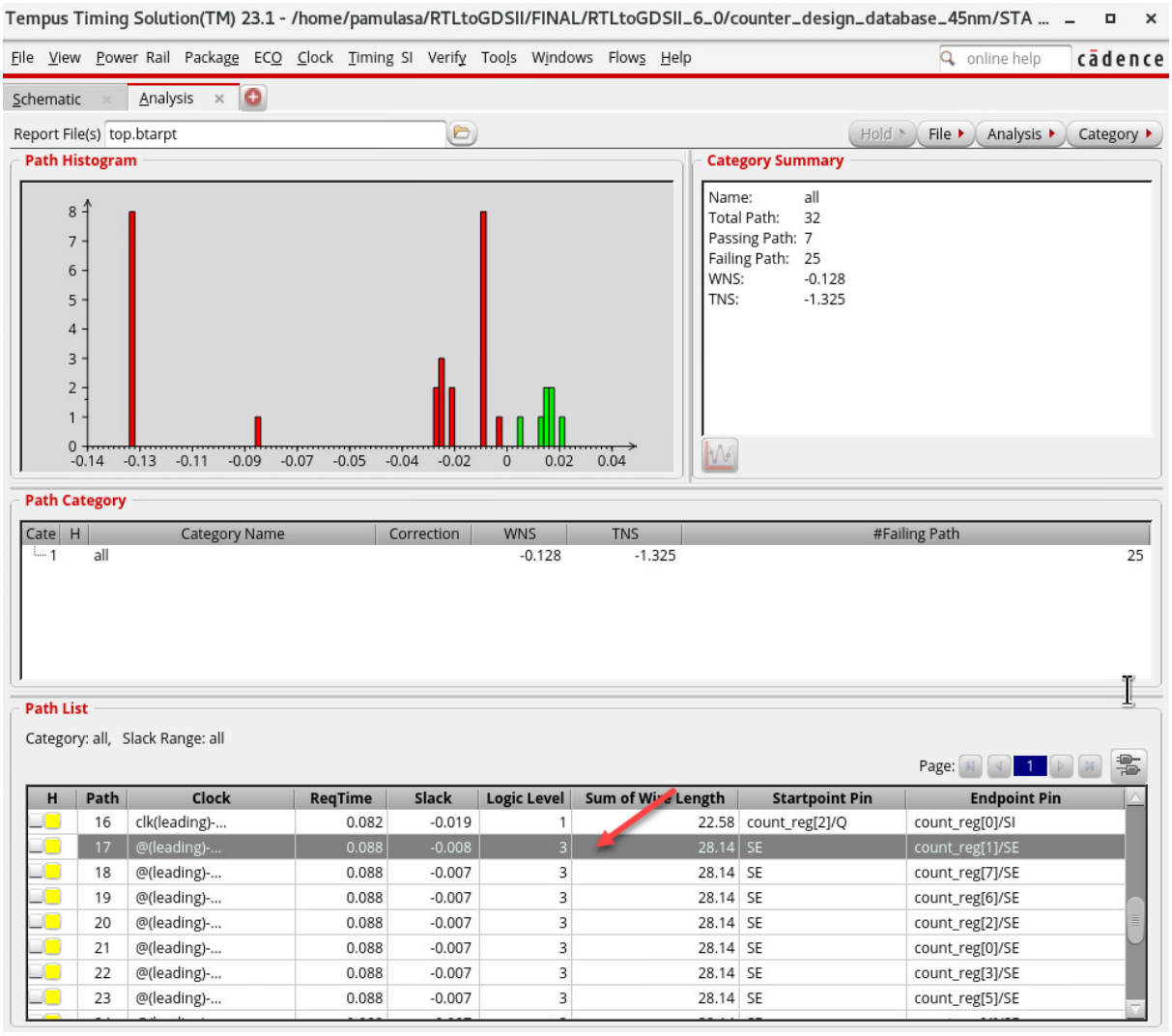
18. Click **Eval all** and choose the buffer cell (BUFX*).

```
###############################################################################
------------------------------------------------------------------------------
------------
Library-cell                              Area     Timing-slack   Worst-Slack-Vie
------------------------------------------------------------------------------
------------
Reference                                 0.0000     -0.0296      bc
CLKBUFX6(buf)                             3.0780     -0.0100      bc
BUFX6(buf)                                3.0780     -0.0099      bc
BUFX3(buf)                                2.0520     -0.0093      bc
CLKBUFX3(buf)                             2.0520     -0.0093      bc
CLKBUFX2(buf)                             1.7100     -0.0078      bc
BUFX2(buf)                                1.7100     -0.0076      bc
CLKBUFX12(buf)                            5.1300     -0.0072      bc
CLKBUFX16(buf)                            6.8400     -0.0072      bc
BUFX12(buf)                               5.1300     -0.0071      bc
BUFX16(buf)                               6.8400     -0.0071      bc
BUFX8(buf)                                3.7620     -0.0070      bc
CLKBUFX8(buf)                             3.7620     -0.0070      bc
BUFX20(buf)                               8.2080     -0.0068      bc
CLKBUFX20(buf)                            8.2080     -0.0068      bc
CLKBUFX4(buf)                             2.3940     -0.0066      bc
BUFX4(buf)                                2.3940     -0.0065      bc
------------------------------------------------------------------------------
```

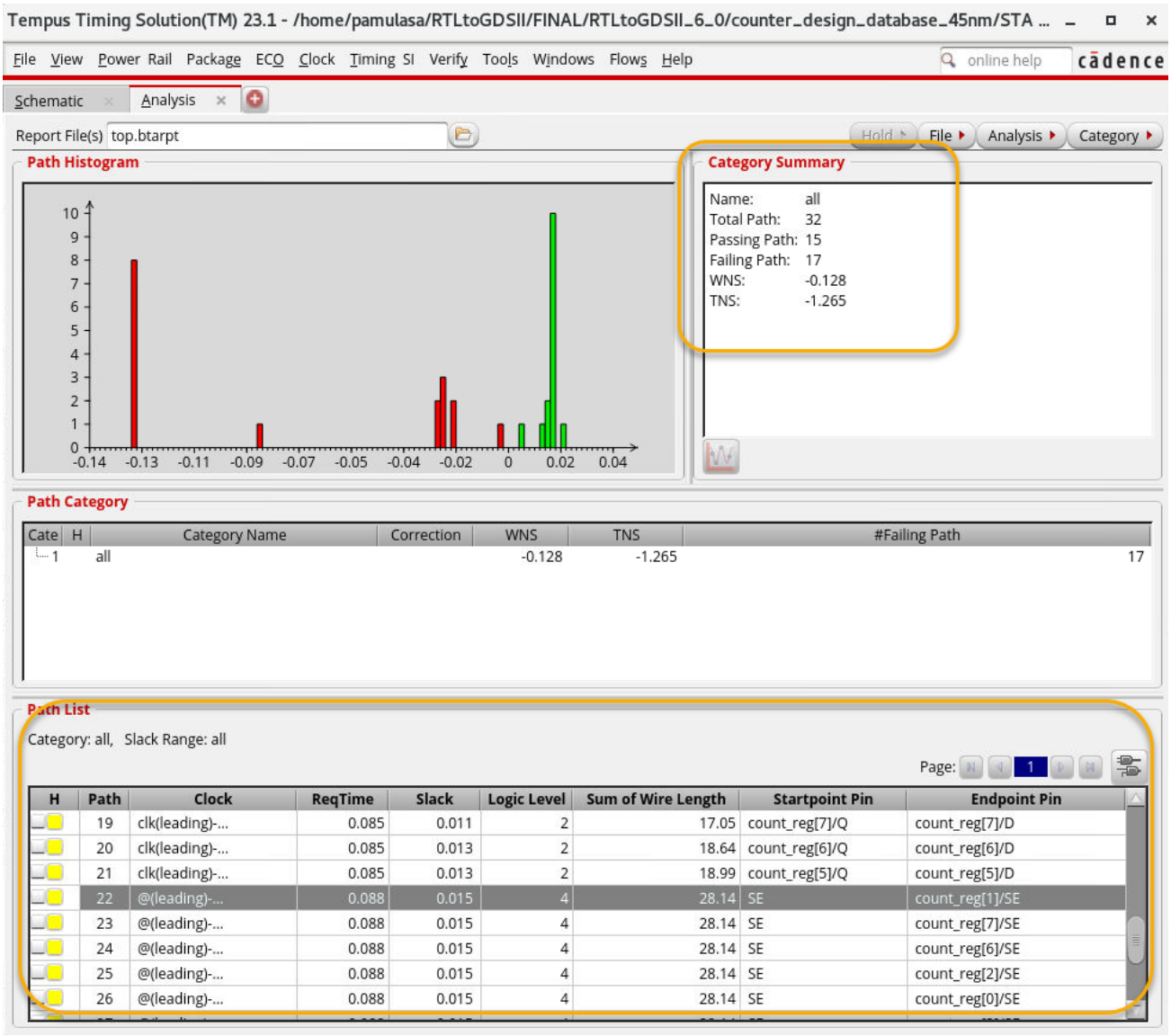
**Note:** Observe that **BUFX4** cell is the best delay cell to add on the path to reduce the timing slack but has the highest area penalty. So, ensure that if you have the area constraint or less area is present in your design, then go for the cell, which adds moderate delay with a lesser area overhead.

So, choose the **BUFX4** cell in this scenario because the area of **BUFX4** is less than the **BUFX4** cell, and also Timing-Slack reduction is not much different between those two cells.

19. After adding the **BUFX4** cell, rerun the **Timing Debug** report and analyze the Hold Slack as shown in **Step 10**



20. Likewise, add more buffers on the same path, analyze the Hold slack for the pin **SE,** and notice the reduction in **WNS.**



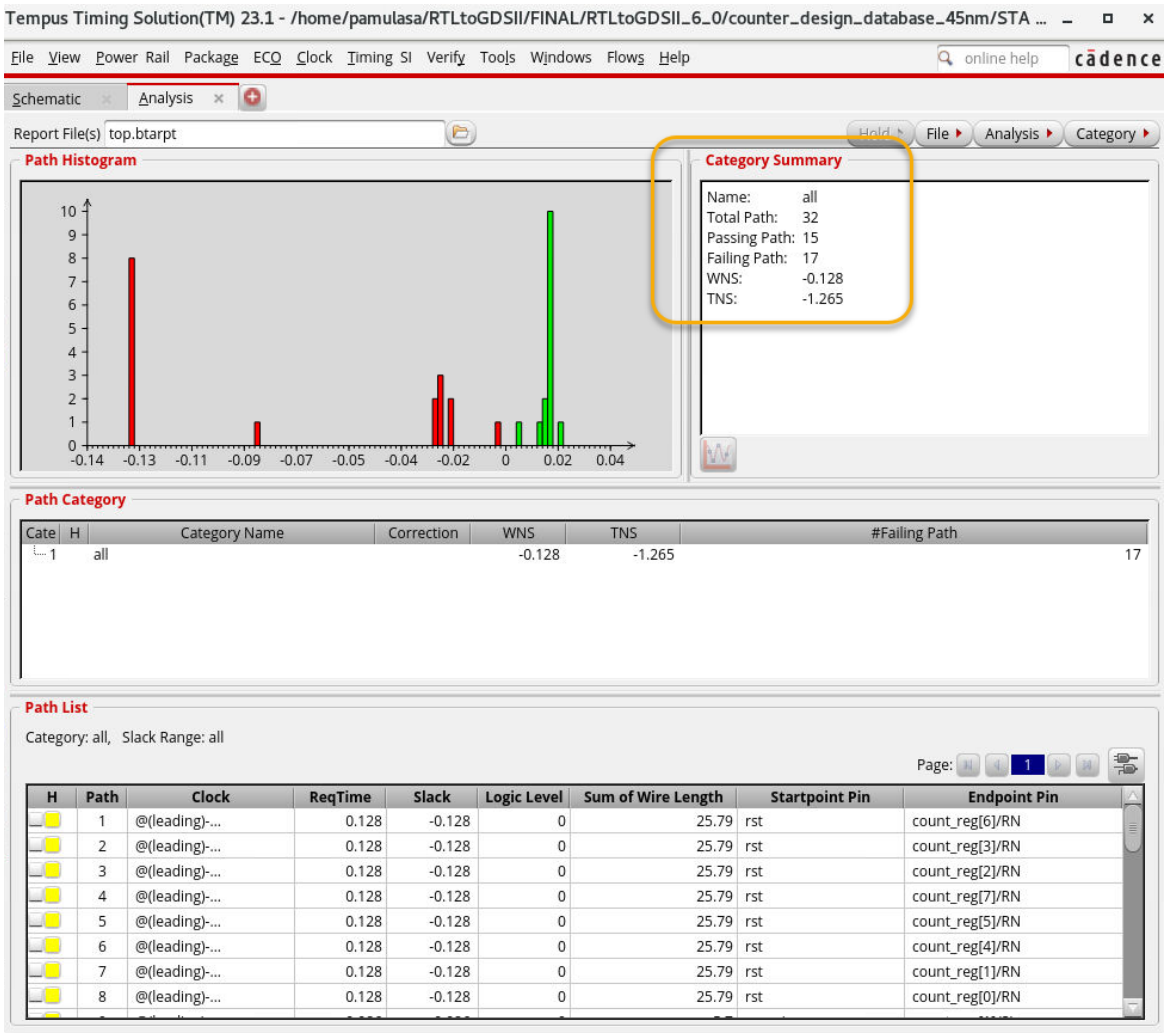
21. Now, check the analysis coverage using the following command to ensure the **Setup Slack** is not affected by the changes made in the Hold Violated Paths.

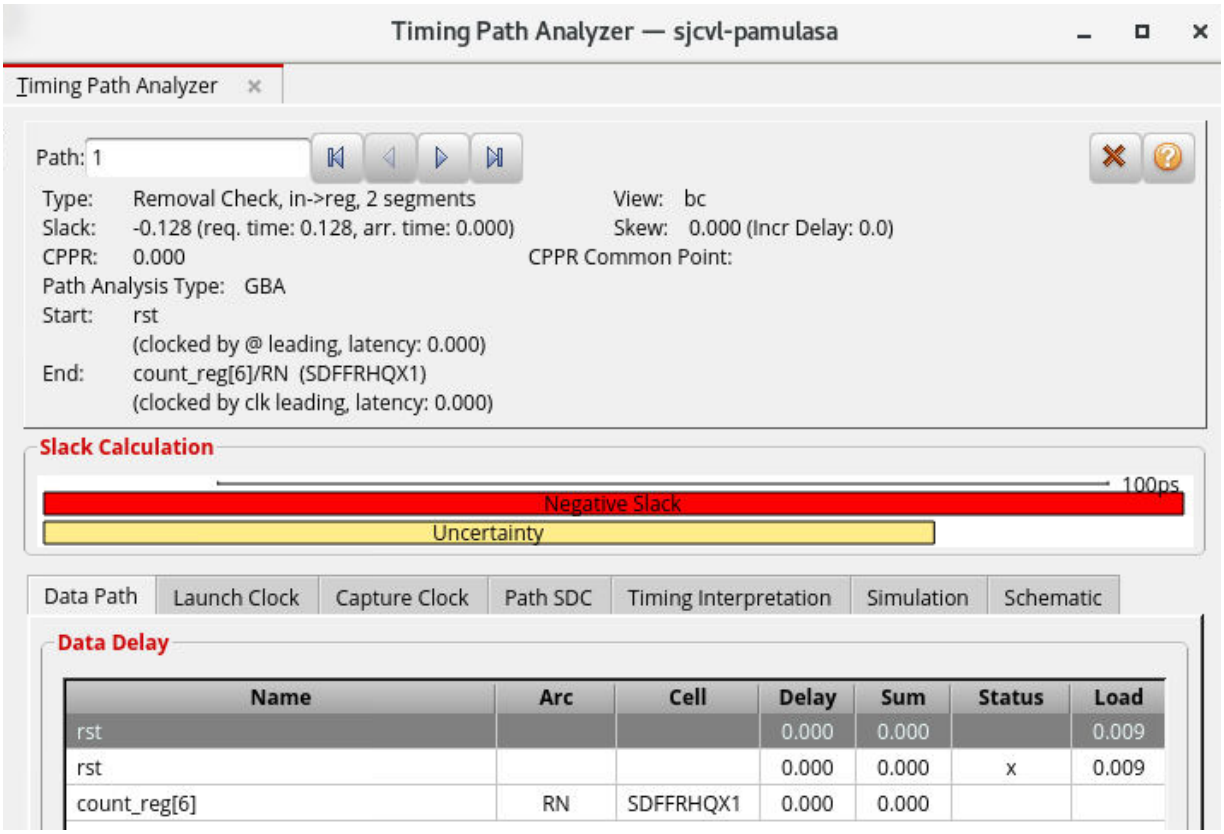
```
report_analysis_coverage
```

```
############################################################
#  Generated by:      Cadence Tempus 23.10-p001_1
#  OS:                Linux x86_64(Host ID sjcvl-pamulasa)
#  Generated on:      Wed Mar 20 02:00:26 2024
#  Design:            counter
#  Command:           report_analysis_coverage
############################################################
    ------------------------------------------------------------------------------------------
                                  TIMING CHECK COVERAGE SUMMARY
    ------------------------------------------------------------------------------------------
     Check Type                     No. of    Met             Violated         Untested
                                    Checks
    ------------------------------------------------------------------------------------------
    ExternalDelay (Late)           8         8 (100%)        0 (0%)           0 (0%)
    Hold                           24        15 (62%)        9 (37%)          0 (0%)
    PulseWidth                     24        16 (66%)        0 (0%)           8 (33%)
    Recovery                       8         8 (100%)        0 (0%)           0 (0%)
    Removal                        8         0 (0%)          8 (100%)         0 (0%)
    Setup                          24        24 (100%)       0 (0%)           0 (0%)
    ------------------------------------------------------------------------------------------
```

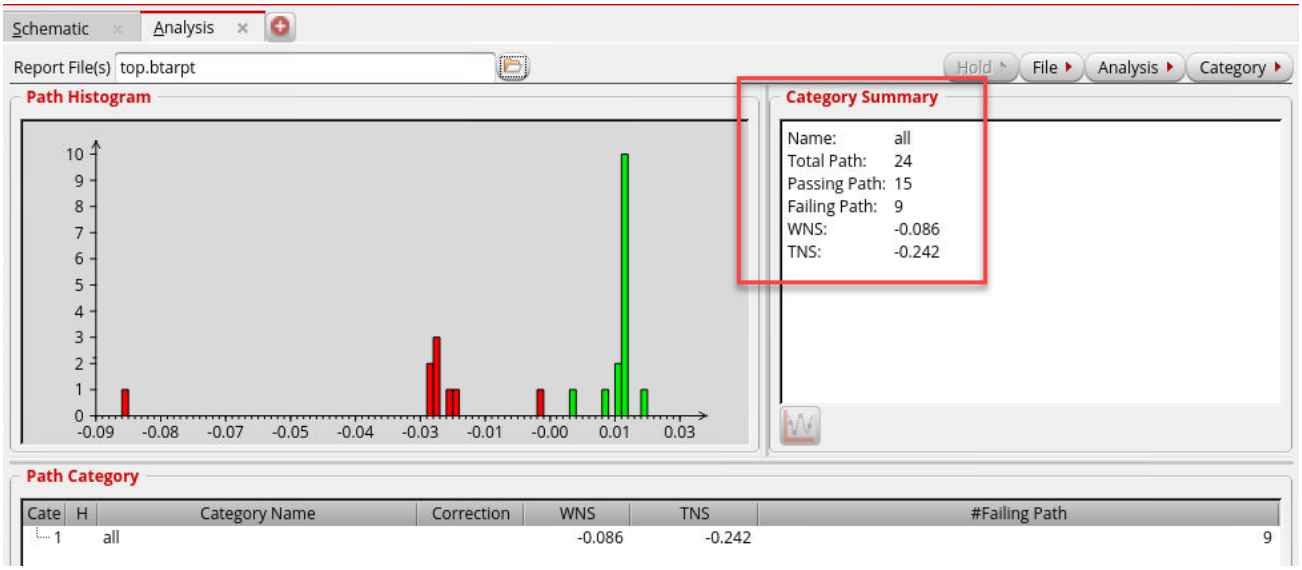
22. Now. Try a similar exercise by buffering the **rst** pin to fix the **Hold Slack** Violations and compare the results.

    Earlier:

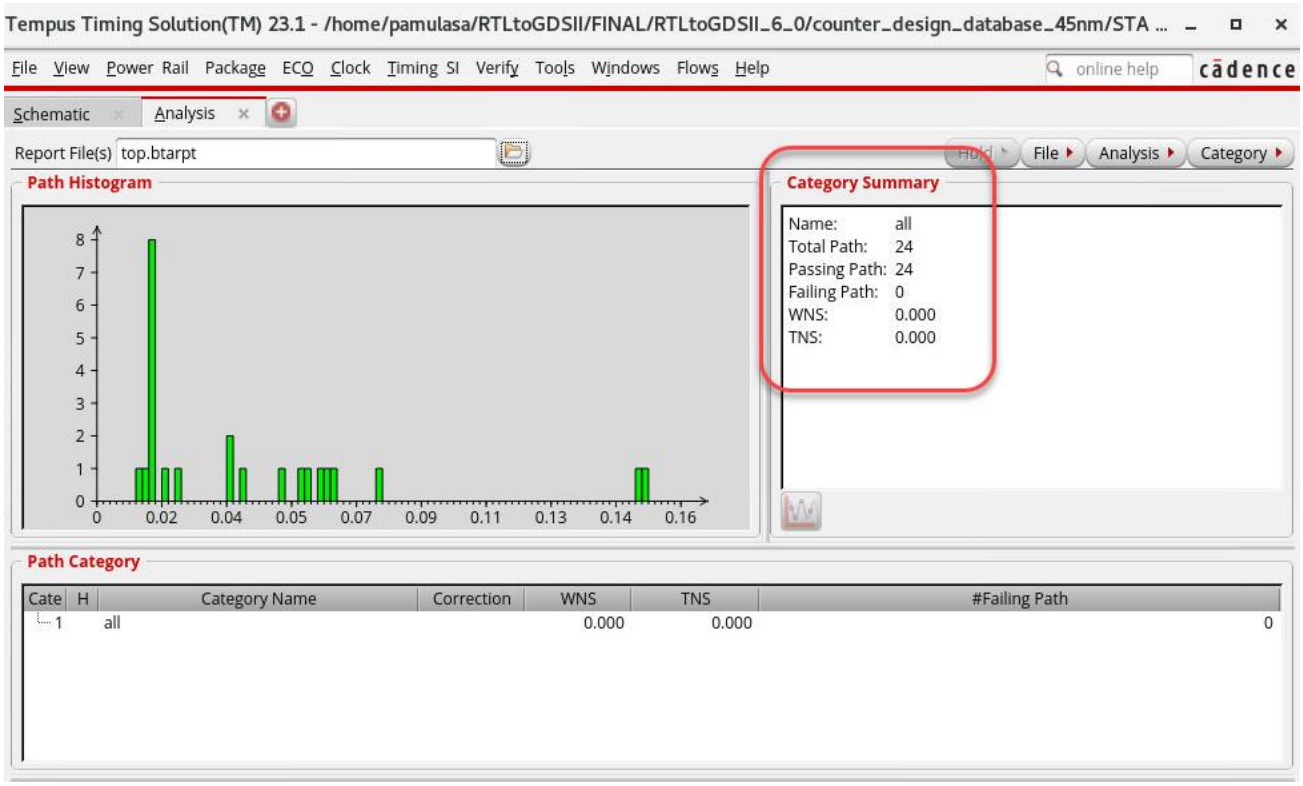





Later:



23. Similarly, fix the remaining timing violations. As a result, it will show zero failing paths in the final timing report, as shown in the figure below.




## Summary

1. You can run timing analysis from within Innovus. You can also evaluate and create timing-fixing ECOs interactively from within Tempus.

2. You can run independent timing analysis from within Tempus. Independent analysis frees up Innovus to do other things.

End of Lab