RTL-to-GDS Flow Tutorial

Course Instructor: Amir Sodagar
TA: Mohsen Namavar

Winter 2024-2025

Contents

1 Introduction to RTL-to-GDS Flow

The **RTL-to-GDSII flow** is the structured process that transforms a high-level hardware description into a manufacturable silicon layout. This step-by-step methodology ensures that a digital circuit, described in a **Register Transfer Level (RTL)** format, is implemented in silicon while meeting power, performance, and area (PPA) constraints. The entire process is driven by Electronic Design Automation (EDA) tools, enabling automation, optimization, and verification before fabrication.

1.1 Why is RTL-to-GDS Flow Necessary?

When designing digital systems, engineers start with a functional model written in `Verilog` or `VHDL`. However, this description is far from a physical implementation. A well-defined RTL-to-GDSII process ensures that:

- The design meets functional correctness through verification.
- Logic gates are synthesized from the RTL description and optimized for timing and power.
- The design follows fabrication constraints, ensuring manufacturability.
- Electrical and physical integrity checks prevent design failures.

Skipping or incorrectly executing any of these steps can result in chips that are **non-functional**, **inefficient**, or even **unmanufacturable**.

1.2 Overview of the RTL-to-GDS Flow

The RTL-to-GDSII flow consists of several key stages, illustrated in Figure 1. Each stage plays a crucial role in transforming the design from an abstract RTL model to a fully verified layout.
The main stages in the flow are:

1. **RTL Design:** The circuit functionality is described using `Verilog` or `VHDL`.
2. **Logic Synthesis:** The RTL is converted into a gate-level representation using a standard cell library, optimizing for performance and area.
3. **Design for Testability (DFT):** Test structures such as scan chains and Built-In Self-Test (BIST) are inserted.
4. **Physical Design (PnR):** The design undergoes multiple steps:
    - **Floorplanning:** Defines the chip layout, pin placement, and power grid.
    - **Placement:** Standard cells are placed on the die.
    - **Clock Tree Synthesis (CTS):** The clock distribution network is generated.
    - **Routing:** Metal interconnections are created.
5. **Signoff and Verification:** This includes:

- **Static Timing Analysis (STA):** Ensures the design meets timing constraints.
- **Design Rule Check (DRC) and Layout vs. Schematic (LVS):** Verifies that the design adheres to manufacturing rules and matches the intended functionality.

6. **Final GDSII Generation:** The layout is converted into a GDSII format, which is sent to the foundry for fabrication.

1.3 Conclusion

The RTL-to-GDSII flow is the essential bridge between high-level hardware design and real-world silicon implementation. This process ensures a design is not only functionally correct but also optimized for fabrication. In the upcoming sections, we will explore each of these steps in greater detail, providing hands-on guidance on implementing a digital circuit from RTL to a final manufacturable GDSII file.

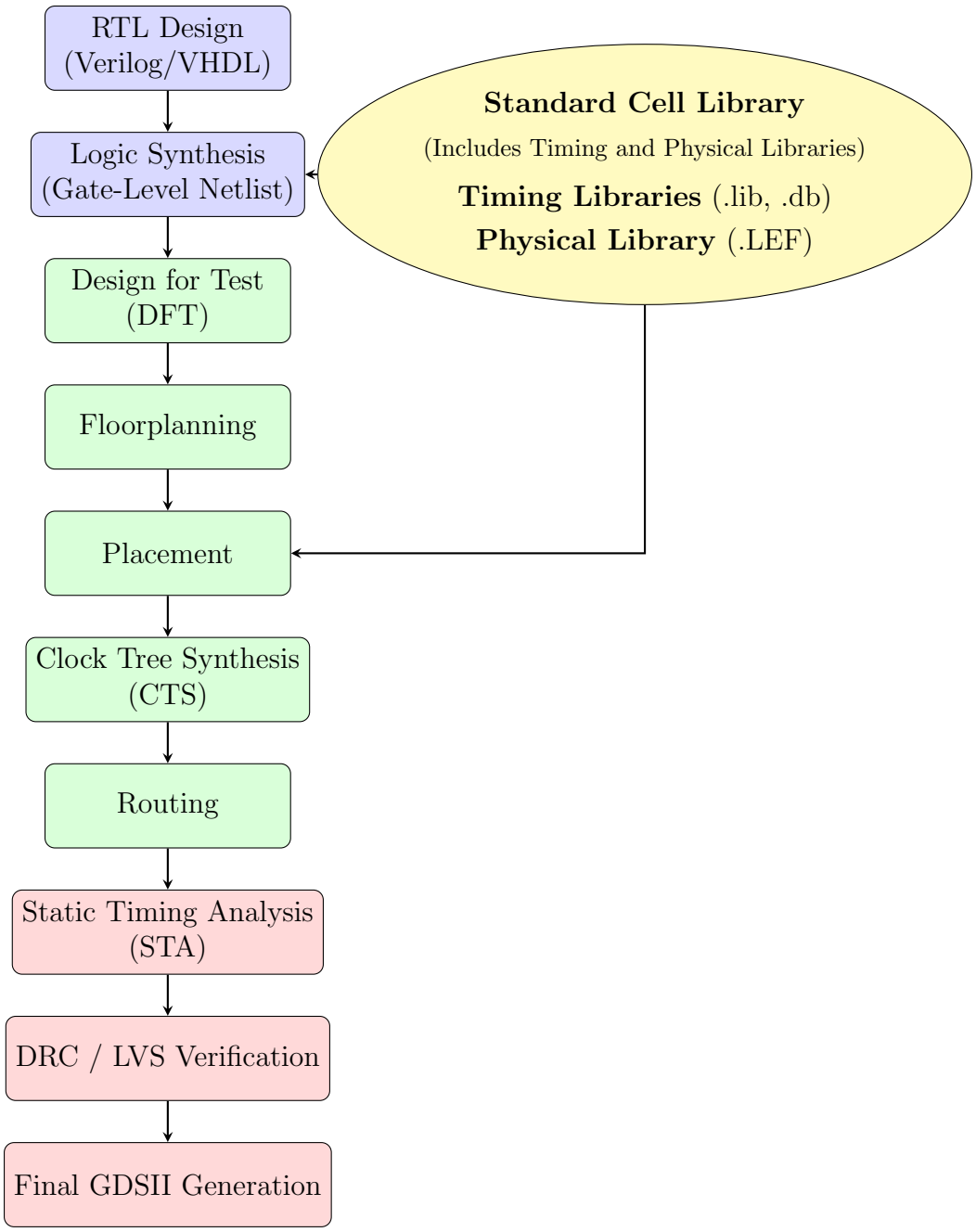

Figure 1: Overview of the RTL-to-GDSII Flow

2 Initialization

Before starting the RTL-to-GDS flow, we need to set up the working environment and initialize the required CAD tools. This section provides step-by-step instructions for setting up the workspace.

2.1 Creating the Working Directory

First, open a terminal and execute the following commands to create the necessary directories:

```
mkdir -p $HOME/VLSI4612
cd $HOME/VLSI4612
mkdir -p project3
cd project3
```

The `project3` folder is your main working directory, where you will store all related files and subdirectories for this project.

2.2 Project Directory Setup and Script Copying

To maintain a well-organized design flow, we structure our project directory as follows:

```
project3/
├── RTL/                  # RTL Design (Verilog/VHDL)
│   ├── src/                  # Source files
│   ├── tb/                   # Testbench files
│
├── Simulation/
│   ├── scripts/               # Simulation scripts
│   ├── results/               # Simulation results
│
├── Synthesis/            # Synthesis using Genus, Yosys, etc.
│   ├── scripts/              # Synthesis scripts (Tcl, Makefiles)
│   ├── reports/              # Timing, Area, Power Reports
│   ├── netlist/              # Synthesized Netlist
│
├── PnR/                  # Place & Route with Innovus, OpenROAD, etc.
│   ├── scripts/              # PnR scripts
│   ├── gds/                  # Final GDSII output
│   ├── reports/              # Timing, DRC, LVS Reports
```

Instead of manually creating directories and copying scripts separately, we have prepared a pre-structured directory on the server that includes:

- **RTL/** - Contains Verilog/VHDL source files and testbenches.
- **Simulation/** - Includes scripts for pre- and post-synthesis simulation.
- **Synthesis/** - Stores TCL scripts for logic synthesis and reports.

- **PnR/** - Holds place-and-route scripts, reports, and final GDSII files.

To set up the project, execute the following command in the terminal:

```
cp -r /cs/course/4612/dir/* $HOME/VLSI4612/project3/
```

This command performs the following:

- Creates all necessary directories under `$HOME/VLSI4612/project3`.
- Copies all pre-prepared scripts into the respective folders.
- Preserves the directory structure from the server to ensure correctness.

2.2.1 Importance of Scripting in the RTL-to-GDS Flow

In the chip design industry, tasks such as simulation, synthesis, and place-and-route (PnR) heavily rely on **automation through scripting** rather than manual GUI-based operations. Using scripts significantly improves the design process by:

- **Reducing repetitive tasks:** Running CAD tools manually can be time-consuming. Scripts automate these tasks, reducing human effort.
- **Ensuring consistency:** Every design step follows the same predefined settings, reducing variability in results.
- **Speeding up execution:** Scripts allow batch processing of tasks without requiring user intervention, improving efficiency.
- **Enhancing reproducibility:** Once a script is written and tested, it can be reused across multiple projects, ensuring a standardized workflow.
- **Providing customization and flexibility:** Scripts allow designers to tweak parameters dynamically without modifying the core flow.

For this reason, in this tutorial, we will utilize **Tcl scripts** for Synthesis and PnR and **Makefiles** for simulation. These scripts are already included in the directory copied from the server.

2.2.2 Verifying the Setup

After running the command, verify that all files and directories have been copied successfully by executing:

```
ls -R $HOME/VLSI4612/project3
```

This will list all subdirectories and files inside `project3`, confirming that everything is in place.

2.2.3 Setting Script Permissions

To ensure all copied scripts are executable, run:

```
find "$HOME/VLSI4612/project3" -type f -name "*.tcl" -exec chmod +x {} \;
find "$HOME/VLSI4612/project3" -type f -name "*.csh" -exec chmod +x {} \;
```

This will:

- Make all shell scripts (`.csh`) executable.
- Make all TCL scripts (`.tcl`) executable for use in tools like Genus and Innovus.

Note: The `-r` option in `cp` ensures that all directories and files are copied recursively, maintaining the original structure.

If these commands execute successfully, your project environment is set up and ready to use.

Important Note: In this tutorial, we assume that you are working on **Project 3**, so the folder created is named `project3`. If you are working on your **Final Project**, you should create a folder named, for example, `finalProject`.

When executing the commands for setting up the working directory and copying the necessary scripts, make sure to replace every occurrence of `"project3"` with `"finalProject"` in the commands.

3 Pre-Synthesis Simulation using Cadence Xcelium

To verify your HDL design, you must simulate your HDL code using an HDL simulator. There are several simulators available, such as **Synopsys VCS**, **ModelSim**, and **Cadence Xcelium**. In this course, we will use **Cadence Xcelium** for simulation.

3.1 Preparing for Simulation

First, copy your related RTL source files into the appropriate directories inside the `RTL` folder. In this tutorial, we use a simple counter design for demonstration purposes, but for your project, you must use the correct RTL files as specified in the project description.

To proceed, copy the following files:

```
cp /cs/course/4612/counter/counter.v $HOME/VLSI4612/project3/RTL/src/
cp /cs/course/4612/counter/tb_counter.v $HOME/VLSI4612/project3/RTL/tb/
```

These commands copy:

- `counter.v` → RTL source file into `RTL/src/`
- `tb_counter.v` → Testbench file into `RTL/tb/`

Next, navigate to the `Simulation` directory:

```
cd $HOME/VLSI4612/project3/Simulation
```

Inside the `Simulation` directory, several essential files exist, including:

- `filelist.f` - Specifies the RTL files for simulation.
- `Makefile` - Controls the simulation process.

3.1.1 Modifying `filelist.f`

To run the simulation, you must tell the Xcelium simulator which RTL files to use. Open `filelist.f` in a text editor:

```
gedit filelist.f
```

Inside `filelist.f`, list all your RTL files (one per line), including the testbench. The modified file should look like this:

```
../RTL/src/counter.v
../RTL/tb/tb_counter.v
```

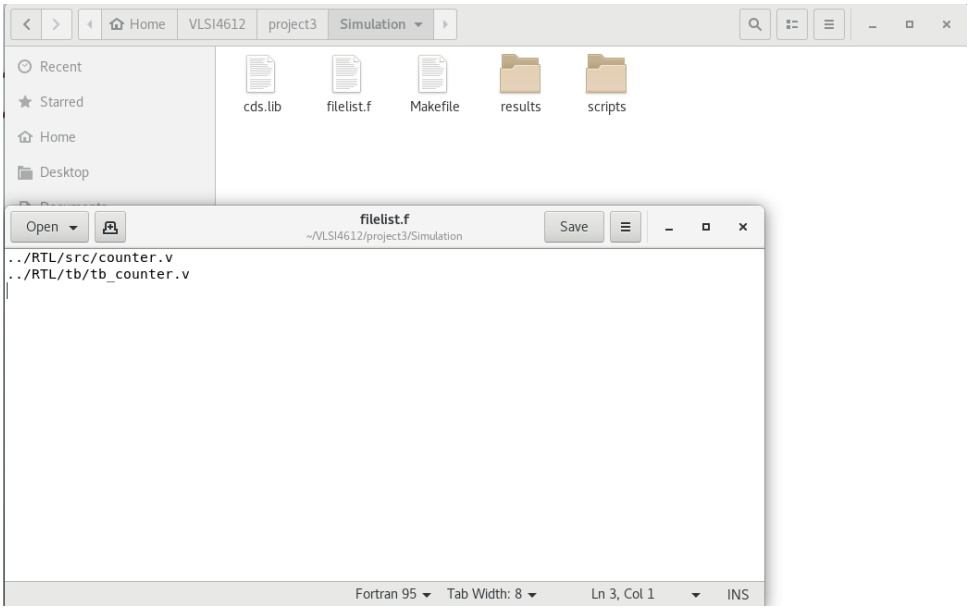

Figure 2: Example of a correctly modified `filelist.f`.

3.1.2 Modifying the Makefile

Next, update the `Makefile` to specify the correct top-level module. Open it using:

```
gedit Makefile
```

Locate the line defining the `TOP_LEVEL` variable and modify it to match the name of your testbench module:

```
TOP_LEVEL = counter_tb
```

Understanding HDL File Names vs. Top Module Names: One common source of confusion for beginners is the distinction between the HDL file name and the top module name inside the file. These are not necessarily the same.

- The **file name** (e.g., `tb_counter.v`) is simply the name of the file where the module is defined.
- The **top module name** is the name used inside the file to define the module using the `module` keyword.

For example, in `tb_counter.v`, the top module is defined as:

```
module counter_tb;
```

However, the file name itself is `tb_counter.v`. When specifying the top-level module in the `Makefile`, you must use the module name (`counter_tb`) rather than the file name.

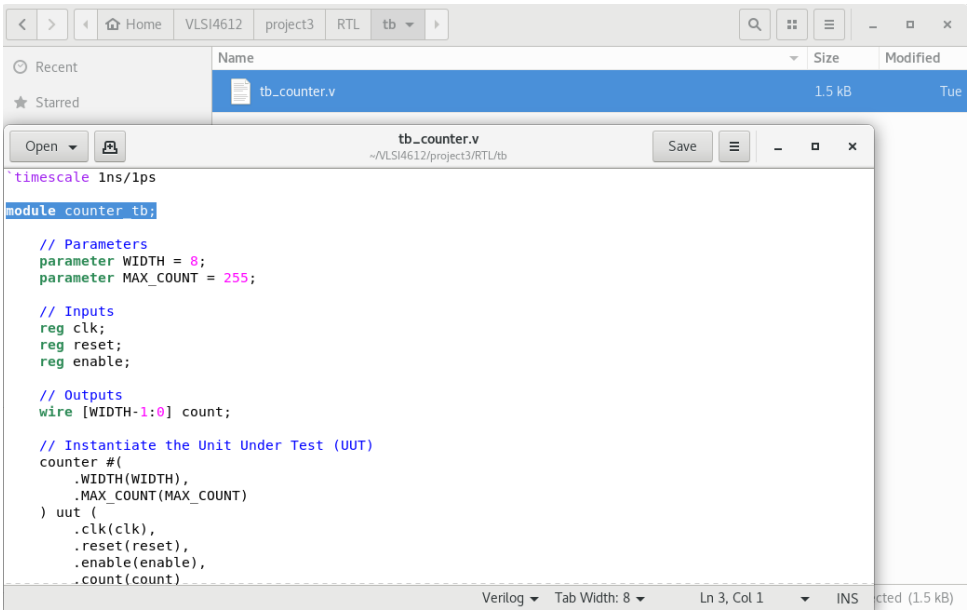

Figure 3: Example of a testbench file (`tb_counter.v`) with its top module name (`counter_tb`).

3.2 Running the Simulation

Before executing the simulation, you must source the required CAD tools to ensure they are available in your environment. Run the following command:

```
vlsi4612
```

This command will initialize the required tools, including Cadence Xcelium. After sourcing the environment, execute the following commands to start the simulation:

```
cd $HOME/VLSI4612/project3/Simulation
make
```

If everything is configured correctly, the **SimVision** window will pop up, displaying the **SimVision Console**, as shown in Figure 4.

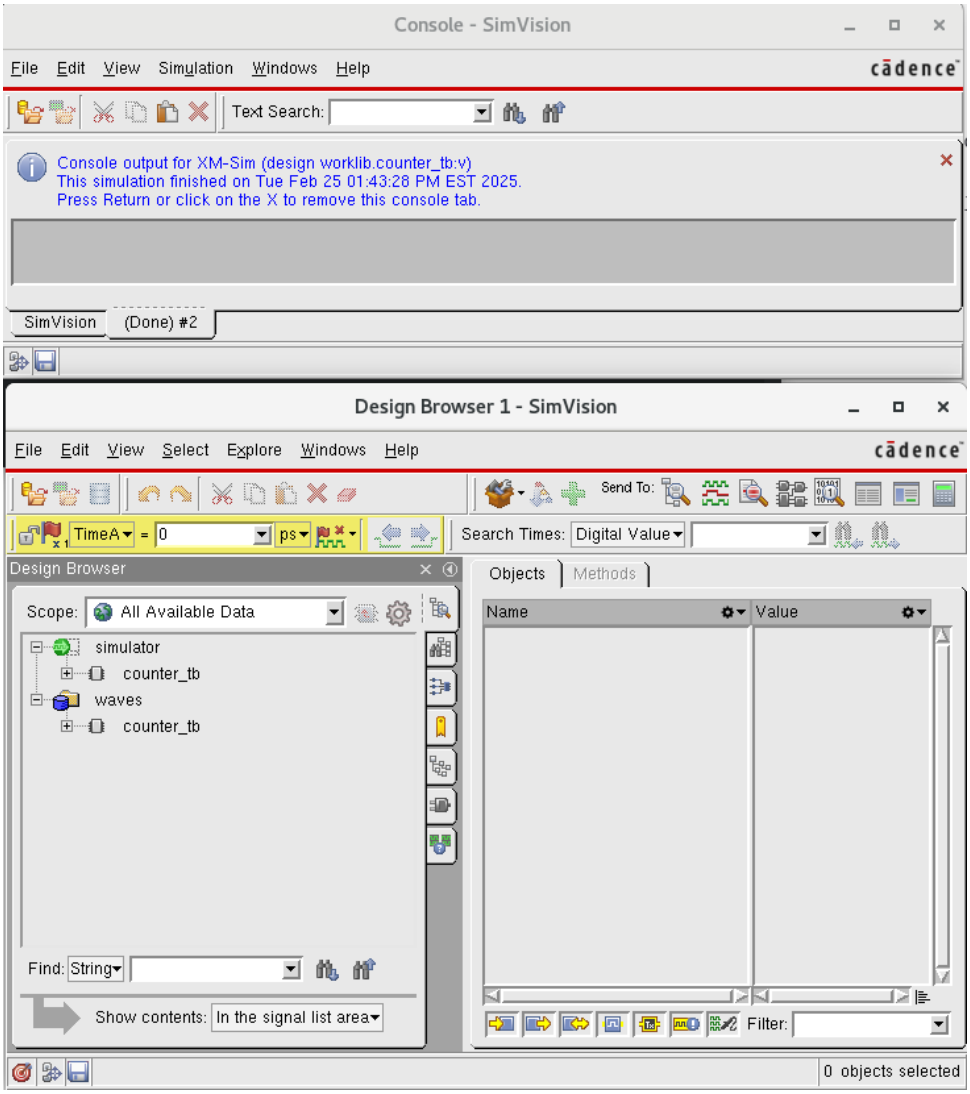

Figure 4: SimVision console window after running the simulation.

3.3 Viewing Waveforms

To analyze the simulation results, you need to plot the waveforms. Follow these steps:

1. In the **SimVision** window, locate your testbench module under the **waves** category in the left panel.
2. Right-click on the testbench module name.
3. Click on **Send to Waveform Window**.

Once completed, you will see the simulation waveforms, as shown in Figure 5.

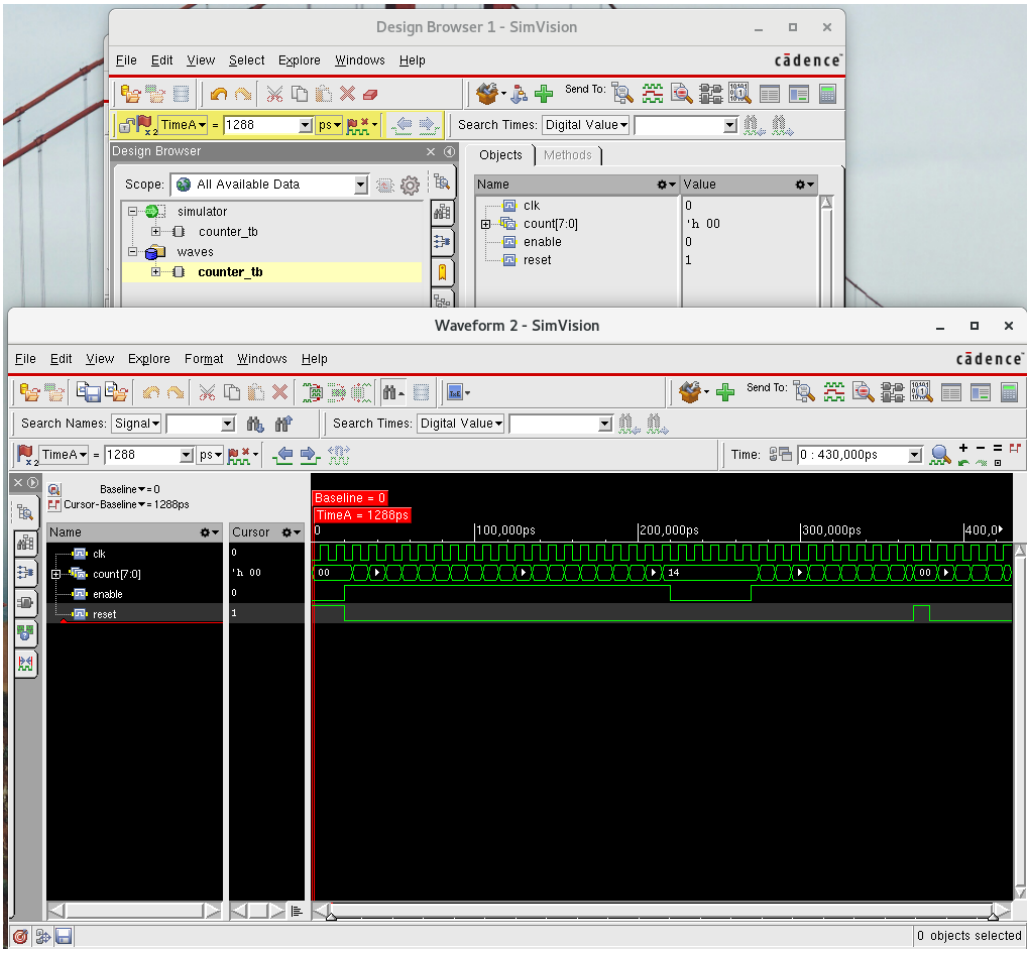

Figure 5: Waveform window in SimVision showing signal transitions.

Verification: After generating the waveforms, analyze the signal transitions and verify that the counter operates correctly. Check the following:

- Ensure that the counter increments correctly on each clock cycle.
- Verify that the reset signal initializes the counter to zero.
- Check that the enable signal correctly controls the counter operation.

If any discrepancies are found, review your RTL code and make necessary modifications.

4 Synthesis

Synthesis is a critical step in digital VLSI design, transforming high-level RTL (Register Transfer Level) descriptions into a gate-level netlist. This netlist consists of interconnected logic gates and sequential elements, selected from a predefined **standard cell library** provided by the fabrication technology.

4.1 Purpose and Importance of Synthesis

The synthesis process plays a key role in bridging the gap between **behavioral design** (RTL) and **physical implementation** (PnR). Its main objectives include:

- **Technology Mapping:** Converts RTL code into standard cells (AND, OR, Flip-Flops, etc.) available in the target semiconductor process.
- **Optimization:** Improves the design based on power, area, and timing constraints to ensure efficient implementation.
- **Timing Closure:** Ensures that the synthesized design meets the required timing specifications by analyzing propagation delays and setup/hold constraints.

4.2 Standard Cell Library and Timing Constraints

To perform synthesis, we need a **standard cell library**, which contains:

- **Pre-characterized logic gates:** NAND, NOR, Flip-Flops, etc.
- **Timing information:** Defines propagation delays, setup/hold times, and constraints for different operating conditions.
- **Power models:** Specifies dynamic and leakage power consumption for each cell.

The timing information is stored in files with extensions `.db` (binary format) or `.lib` (human-readable format). For this tutorial, the required timing library can be found in the following path:

```
/CMC/kits/cadence/GPDK045/gsclib045_all_v4.4/gsclib045/timing
```

The synthesis tool reads these files to ensure that the generated netlist meets the required timing constraints.

4.3 Inputs and Outputs of Synthesis

A synthesis tool takes the following as **inputs**:

- **RTL Source Code:** Describes the hardware functionality in Verilog or VHDL.
- **Standard Cell Library:** Provides logic gate implementations, delays, and constraints.
- **Synthesis Constraints (SDC File):** Defines the clock period, input/output delays, and design constraints.

The synthesis tool generates the following **outputs**:

- **Gate-Level Netlist (`.v`):** A structural Verilog file mapping RTL logic to standard cells.
- **Timing Reports:** Analyze whether the synthesized circuit meets setup and hold time constraints.
- **Power and Area Reports:** Provide estimates of power consumption and chip area usage.
- **Synthesis Constraints (`.sdc` File):** Updated constraints for place-and-route tools.

4.4 Starting the Synthesis Process

Before running synthesis, **close all SimVision windows** from the previous simulation step to avoid conflicts.

Follow these steps:

1. Change the current directory to the synthesis folder:

```
cd $HOME/VLSI4612/project3/Synthesis
```

2. **Re-source the CAD tools if needed:** If a new terminal is opened (for example, if the current terminal was accidentally closed), run:

```
vlsi4612
```

4.5 Reading and Editing the Synthesis Script

Before starting the synthesis process, we need to configure the synthesis script. The synthesis flow is controlled using a **Tcl script** that defines constraints, specifies synthesis settings, and automates the logic optimization process.

To open the synthesis script, execute the following command:

```
gedit scripts/genus_script.tcl
```

This script contains several project-dependent parameters that must be updated according to your design. The key parameters that need modification are:

- **RTL Files:** The variable `set rtlFiles` should include all the Verilog source files needed for synthesis. If your design consists of multiple modules, list all required files separated by spaces.
- **Top-Level Module Name:** The variable `set topModule` must match the name of your design's highest-level module.

Note: There is no need to modify the clock frequency, clock name, or timing constraints in this step.

Standard Cell Library Path: The synthesis script must also reference the technology's standard cell timing library. Ensure that the script correctly points to the following directory:

```
/CMC/kits/cadence/GPDK045/gsclib045_all_v4.4/gsclib045/timing
```

This directory contains the necessary `.lib` and `.db` files required for synthesis. Make sure the `set_db init_lib_search_path` variable in the script is set correctly.

An example of the `genus_script.tcl` file with these modifications is shown in Figure 6.

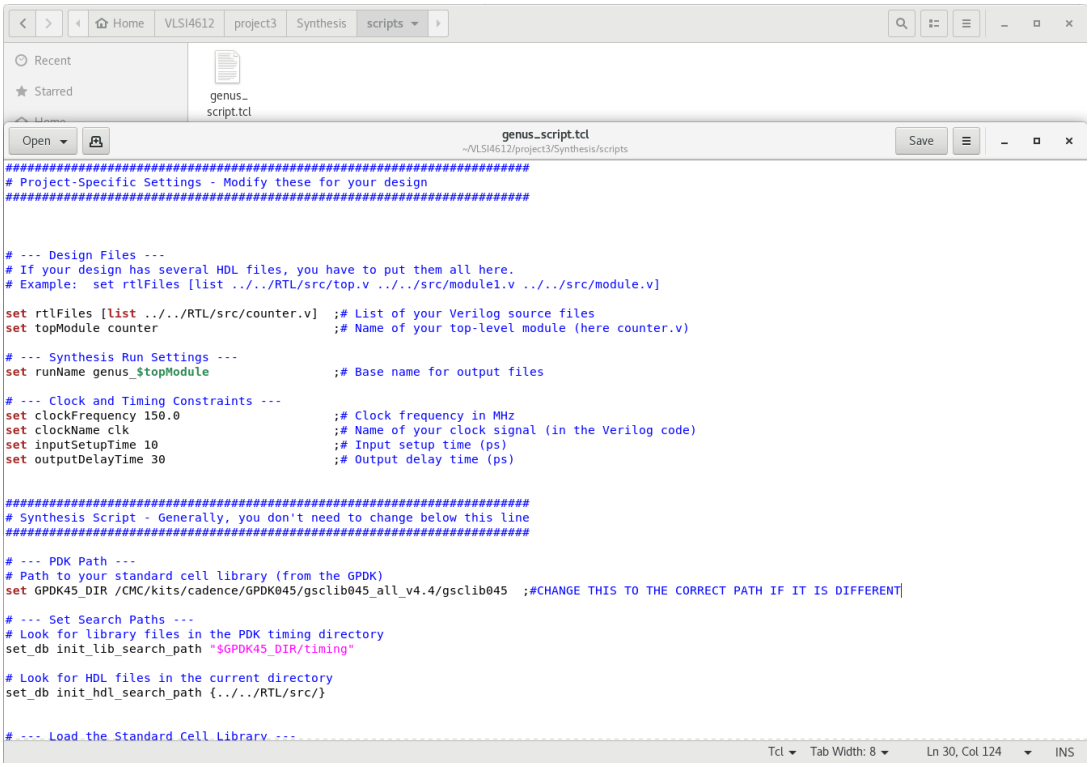

Figure 6: Editing `genus_script.tcl` to specify RTL files, top module, and standard cell library path.

4.6 Running the Synthesis

To start the synthesis process, execute the following commands in the terminal:

```
genus
source scripts/genus_script.tcl
```

If everything is set up correctly, the synthesis process will begin, and the **Genus GUI** will appear, as shown in Figure 7. To view the synthesized schematic, right-click on the **Hier Cell** in the left panel of the Genus GUI, then select **Schematic View**.

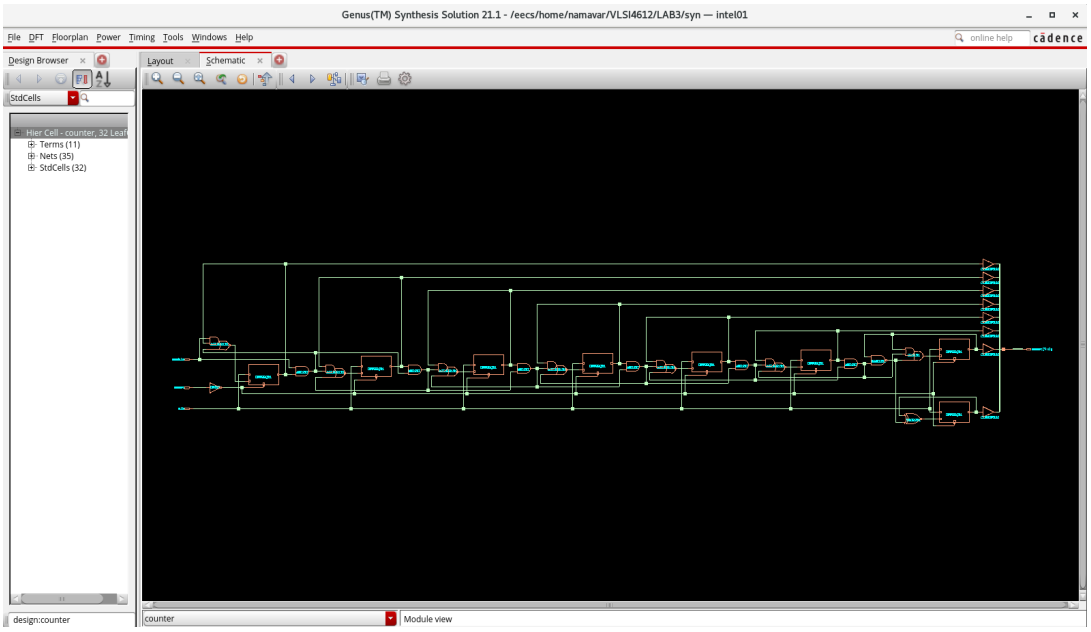

Figure 7: Genus GUI after synthesis completion, showing the synthesized schematic.

4.7 Verifying Synthesis Reports

After synthesis is completed, Genus generates several reports that provide insights into the area, power, and other characteristics of the synthesized design. These reports are stored in the `Synthesis/reports` directory.

Post-Synthesis Area and Power Analysis: Students must check and report the following:

- **Post-Synthesis Area:** Found in the area report file (`*.area.rep`). The unit of area is μm^2.
- **Post-Synthesis Power:** Found in the power report file (`*.power.rep`). The unit of power is *Watts (W)*.

An example of area and power reports is shown in Figure 8.

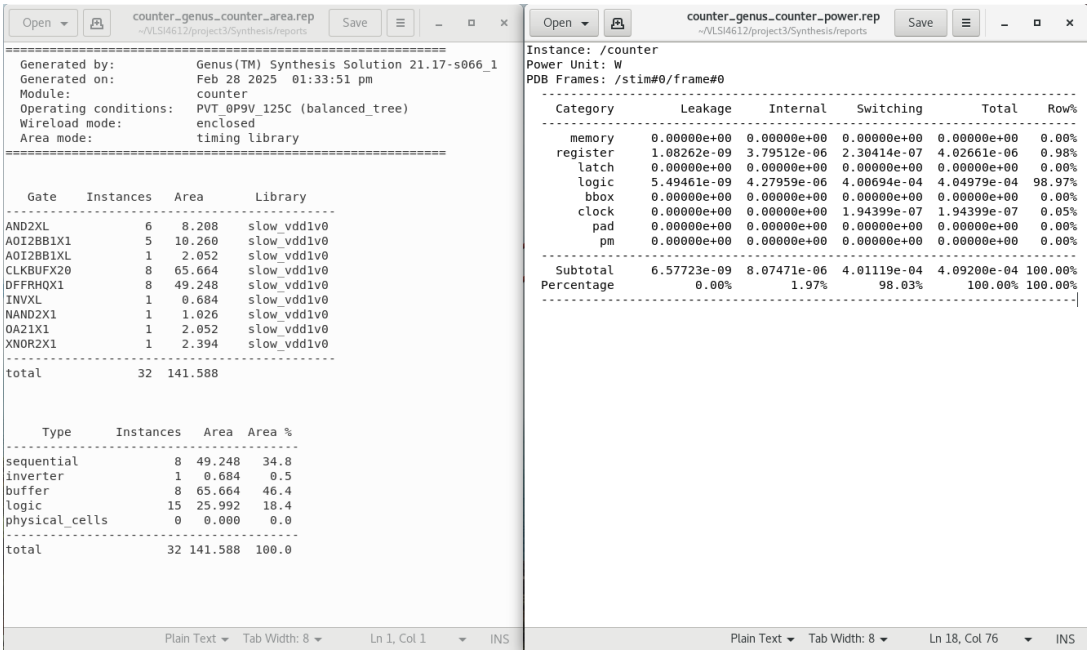

Figure 8: Example of post-synthesis area and power reports generated by Genus.

Additionally, students can check the synthesized netlist, which is stored in the `Synthesis/netlist` directory. This netlist represents the mapped logic gates after synthesis and will be used in the place-and-route (PnR) stage.

Task: Report the post-synthesis area and power values for your specific project.

5 Automatic Layout Generation (Place and Route) Using Cadence Innovus

The **Place and Route (PnR)** process is a crucial step in the VLSI design flow that transforms a synthesized netlist into a fully routed layout, ready for fabrication. This process involves multiple stages, including **floorplanning, standard cell placement, power planning, clock tree synthesis (CTS), routing, and design rule verification**. The goal of PnR is to ensure that all components are optimally placed and interconnected while meeting timing, power, and area constraints.

Modern chip designs require handling millions of standard cells and complex routing, making **manual layout impractical**. Instead, automated layout generation is performed using specialized Electronic Design Automation (EDA) tools such as **Cadence Innovus**. These tools leverage **advanced algorithms and scripting** to efficiently generate high-performance, manufacturable layouts while minimizing design effort and human error.

5.1 Launching Innovus

Before launching Innovus, ensure that all **Genus-related windows are closed**, and you have exited the Genus shell to prevent conflicts.

Next, change the current directory to the `PnR` directory:

```
cd $HOME/VLSI4612/project3/PnR
```

Then, start Innovus by executing:

```
innovus
```

It may take some time for Innovus to load. Once it has finished initializing, the **Innovus GUI** will appear, ready for the Place and Route (PnR) process.

5.2 Importing the Design

The first step in the Place and Route (PnR) process is to import the synthesized design into Innovus. This involves loading the synthesized netlist, technology files, and various design constraints.

1. In the **Innovus** menu bar, click **File** → **Import Design**.
2. The **Design Import** window will appear, as shown in Figure 9.

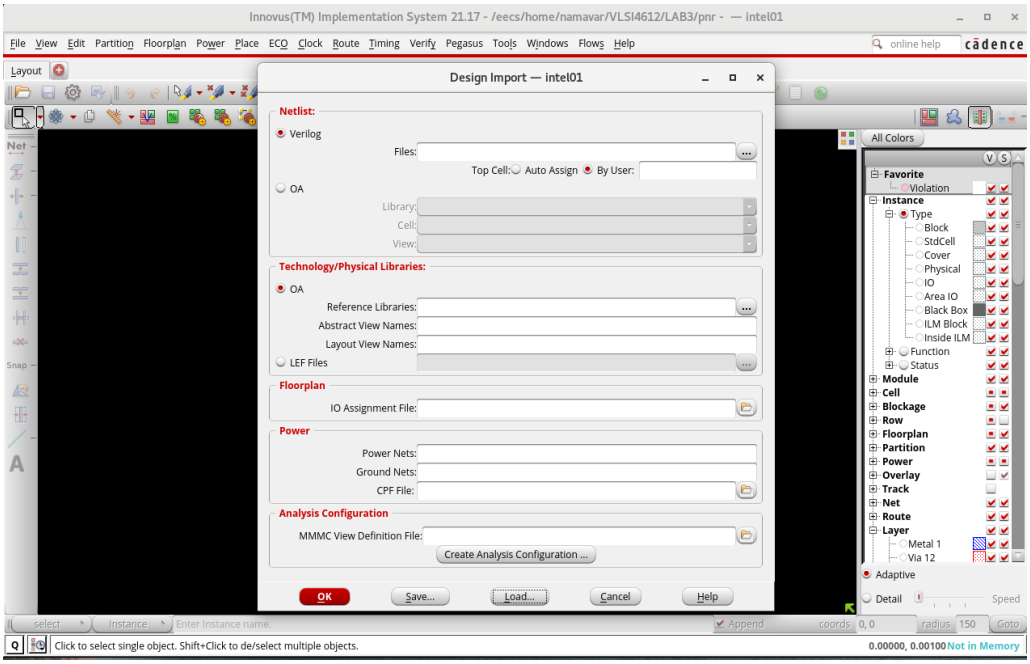

Figure 9: Design Import window in Cadence Innovus.

In this window, you need to specify the following:

- **Netlist:** The output netlist generated by **Genus** after synthesis.
- **LEF Files (Library Exchange Format):** These files define the physical and electrical properties of standard cells, macros, and I/O pads in the given technology. LEF files include:
  - **Cell dimensions**: Width, height, and placement grid.
  - **Pin locations and routing obstructions**: Essential for routing and connectivity.
  - **Metal layer definitions**: Specifies which layers can be used for routing.

  LEF files do not contain transistor-level details but provide enough information for PnR tools to place and route standard cells efficiently.
- **VDD/VSS Net Names:** Defines the power and ground connections for proper power distribution.
- **Analysis Configuration:** Used for timing, signal integrity, and power analysis.

While it is possible to manually configure these settings through the GUI, this approach is time-consuming and error-prone. To maintain consistency and efficiency, we use a pre-defined **initialization script**, which automatically loads all required settings.

5.3 Using the Initialization Script

Instead of manually entering design details each time, we use a pre-written initialization script (`init.globals`). This script ensures that all design files, constraints, and settings are correctly configured before the Place and Route (PnR) process begins.
The initialization script is located in the `PnR/scripts` directory. Before using it, you need to make a few modifications to correctly reference your synthesized design files.

5.3.1 Modifying `init.globals`

Open the script for editing:

```
gedit $HOME/VLSI4612/project3/PnR/scripts/init.globals
```

Inside this script, you must update the references to the synthesized netlist and constraint files. Locate the following lines:

```
# Set Verilog and SDC files
set verilog_file "$netlist_dir/genus_counter.v"
set sdc_file "$netlist_dir/genus_counter.sdc"

# Initialize the Verilog file
set init_verilog "$netlist_dir/genus_counter.v"
```

These lines indicate the output **Verilog netlist** and **SDC constraint file** generated during synthesis. To ensure that the correct files are referenced:

1. Navigate to the synthesis netlist directory:

   ```
   cd $HOME/VLSI4612/project3/Synthesis/netlist
   ```

2. Identify the actual netlist and SDC file names.

3. Replace `genus_counter.v` and `genus_counter.sdc` with the correct filenames.

Additionally, locate the following line in the script:

```
set init_top_cell counter
```

This line defines the top module of your design. Replace `counter` with the actual top module name from your HDL design (not test-bench) to ensure correct initialization.

5.3.2 Modifying `mmmc.view`

Another file that needs modification is the **MMMC (Multi-Mode Multi-Corner)** configuration file, which defines constraint modes and analysis conditions. Open it using:

```
gedit $HOME/VLSI4612/project3/PnR/scripts/mmmc.view
```

Locate and update the following line:

```
#=============================================================
# Constraint Mode Definition
#=============================================================
create_constraint_mode -name constraints -sdc_files "$netlist_dir/genus_counter.sdc"
```

Similar to the `init.globals` file, update this line to reference the correct **SDC file** generated during synthesis.

5.3.3 Loading the Initialization Script in Innovus

Once the necessary modifications are complete, load the script into Innovus:

1. In the **Design Import** window, click the **Load** button.
2. Navigate to the `PnR/scripts` directory and select `init.globals`, as shown in Figure 10.
3. Click **Open** to apply the settings.

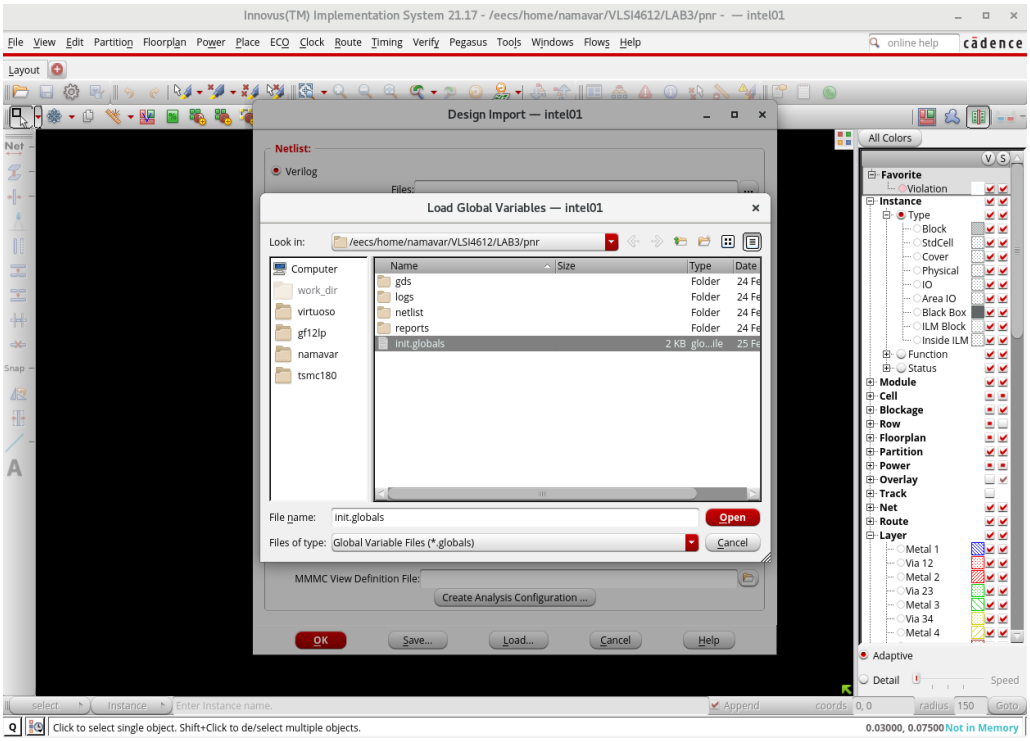

Figure 10: Loading the `init.globals` file in Innovus.

Once loaded, all required design settings, including **netlist files, constraints, and technology settings**, will be automatically configured, ensuring a consistent and error-free design flow.

5.4 Floorplanning

Floorplanning defines the physical dimensions and placement of core and I/O components. To specify the floorplan:

1. Click **Floorplan** → **Specify Floorplan**.
2. In the **Specify Floorplan** window, set **Core Margins by** to **Core to IO Boundary**.
3. Enter **5** (unit: µm) for **Core to Left, Right, Top, and Bottom**.
4. Click **OK**.

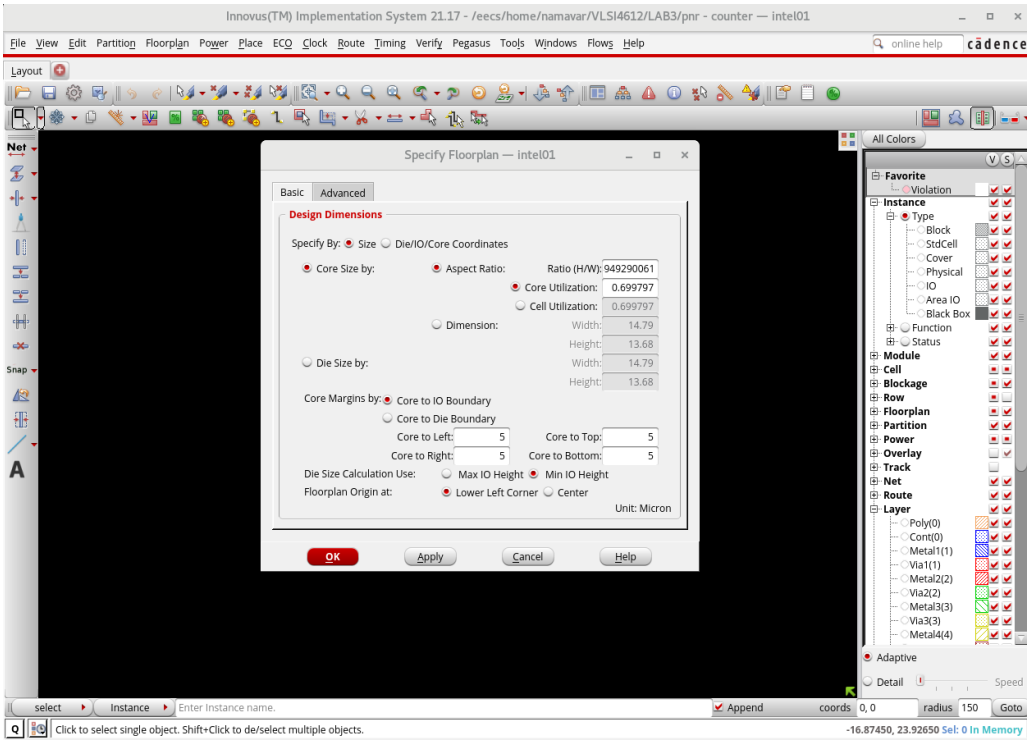

Figure 11: Specify Floorplan window in Innovus.

5.5 Power Planning

After floorplanning, the next step is to set up power distribution for the design.

1. Click **Power** → **Power Planning** → **Add Ring**.
2. In the **Add Ring** window:
   - Type **VDD VSS** in the **Net(s)** field.
   - In the **Ring Configuration** section, enter **1** µm for **Width, Spacing, and Offset**.
3. Click **OK**.

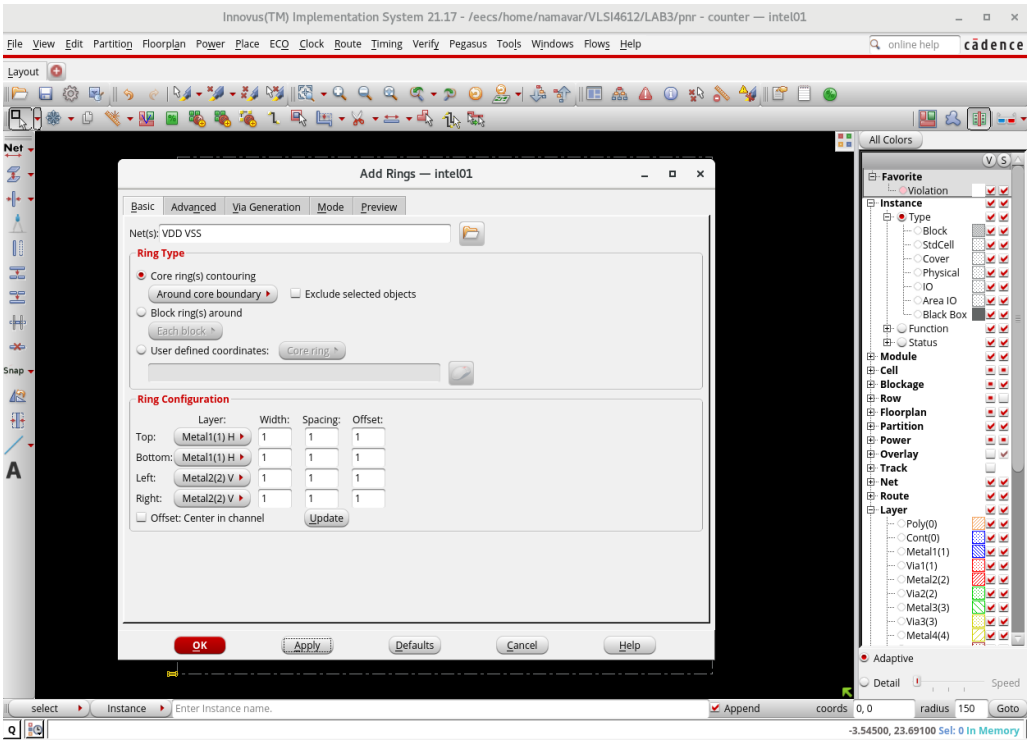

Figure 12: Add Ring window in Innovus.

If the configuration is correct, you should see power rings surrounding the core, as shown in Figure 13.

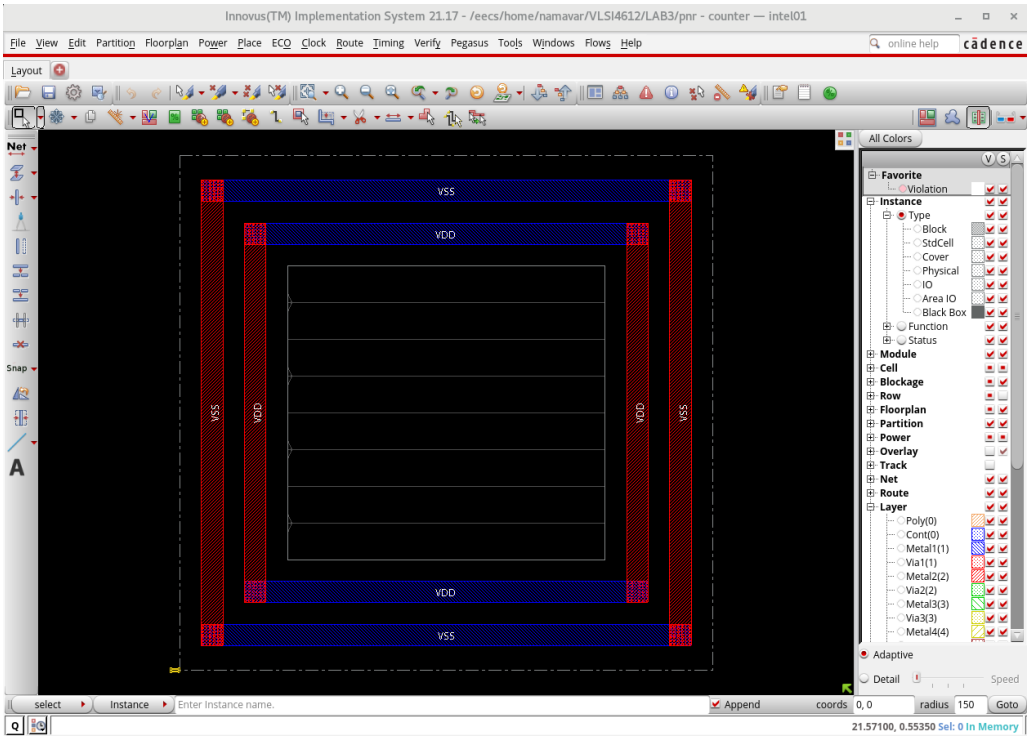

Figure 13: Power rings after adding VDD and VSS rings.

5.6 Special Routing (Power and Ground Routing)

To perform power and ground (VDD/VSS) routing, follow these steps:

1. Click **Route** → **Special Route**.
2. In the **Net(s)** field, type **VDD VSS**.
3. Click **OK**.

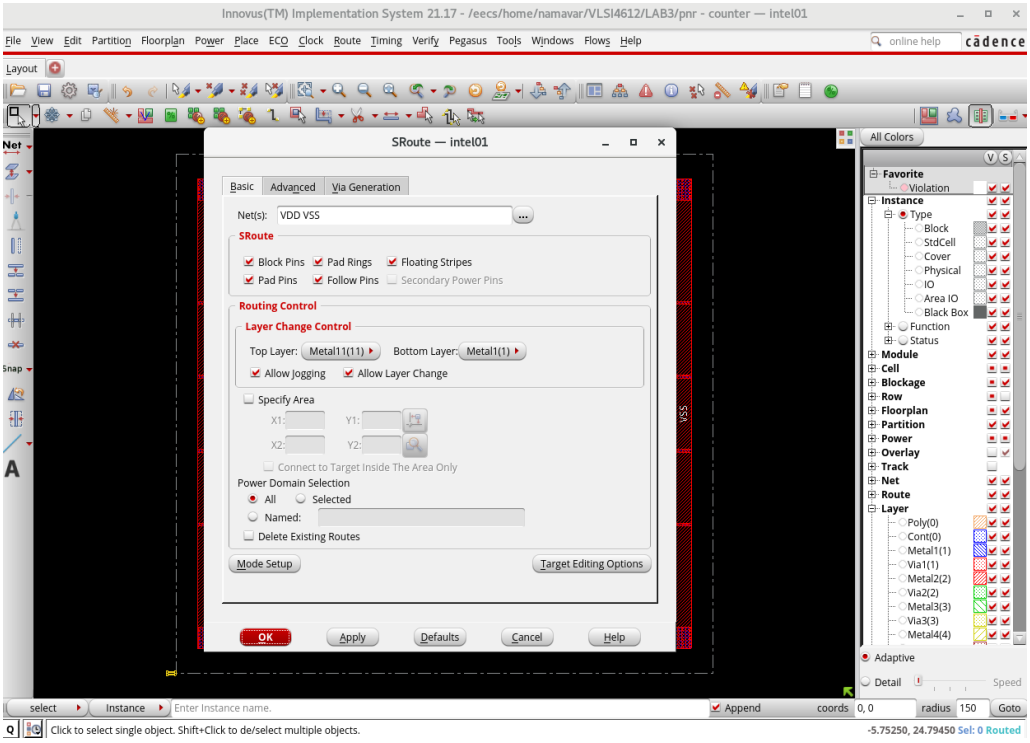

Figure 14: Special Route window for power and ground routing.

Once routing is complete, the power and ground connections will be routed as shown in Figure 15.

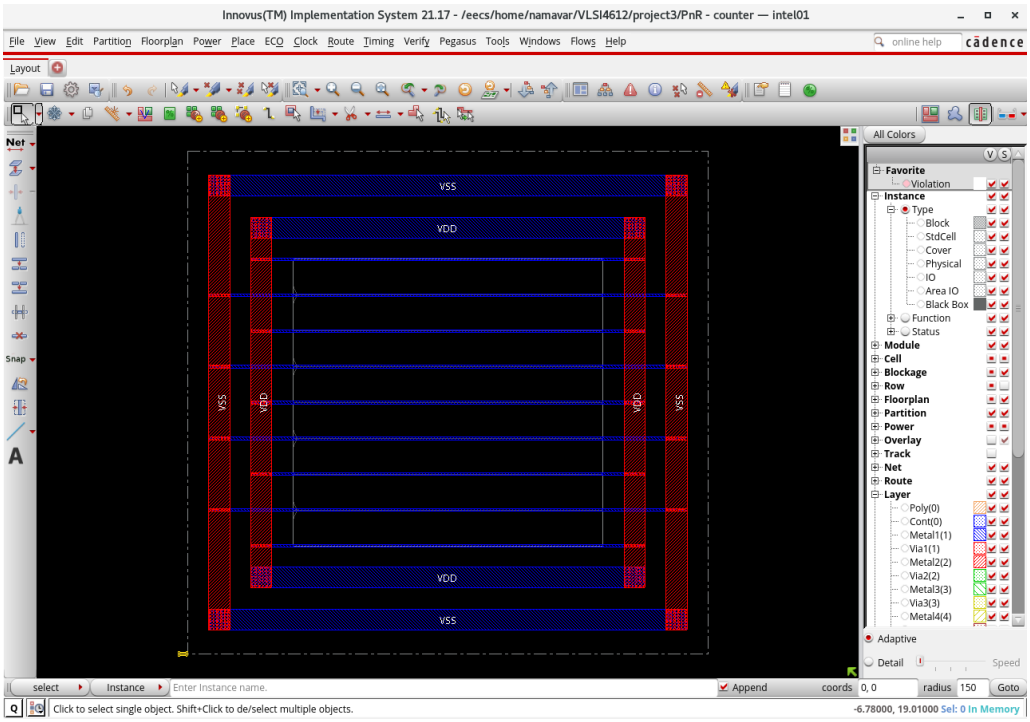

Figure 15: Power and ground routing result after special routing.

5.7 Placement

After defining the floorplan and power rings, the next step is to place the standard cells.

1. Click **Place** → **Place Standard Cells**.
2. The **Place** window will appear, as shown in Figure 16.
3. Click on the **Mode...** button in the **Place** window.
4. In the **Placement Mode** window (Figure 17), click **Set Defaults**, then press **OK**.
5. Finally, press **OK** in the **Place** window to start cell placement.

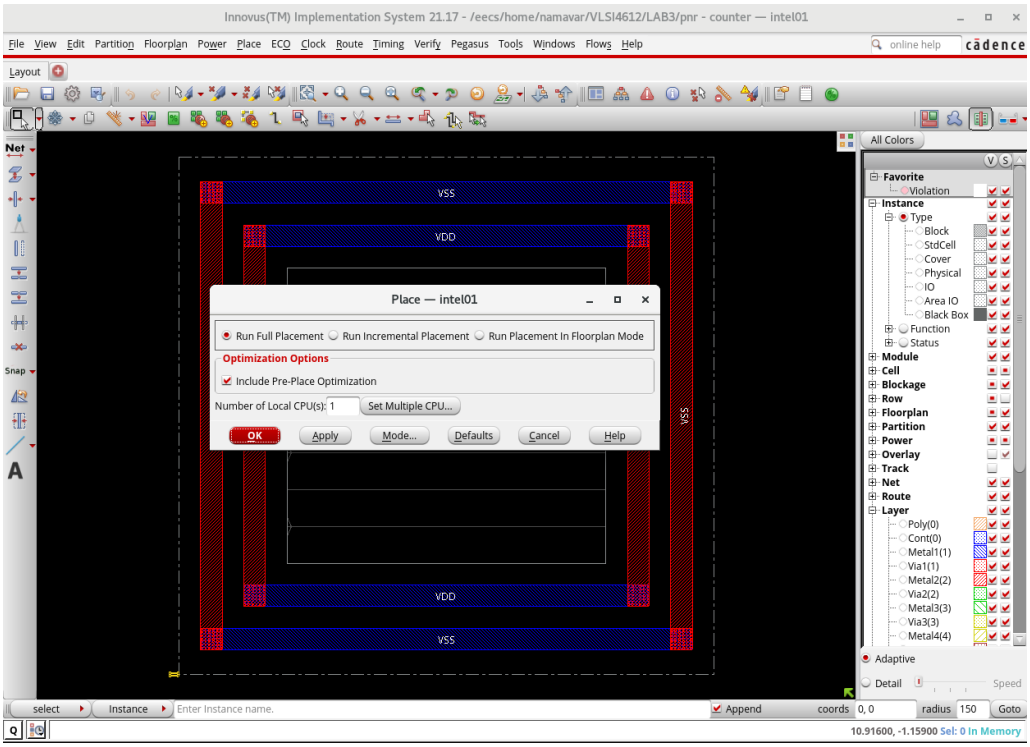

Figure 16: Place Standard Cells window in Innovus.

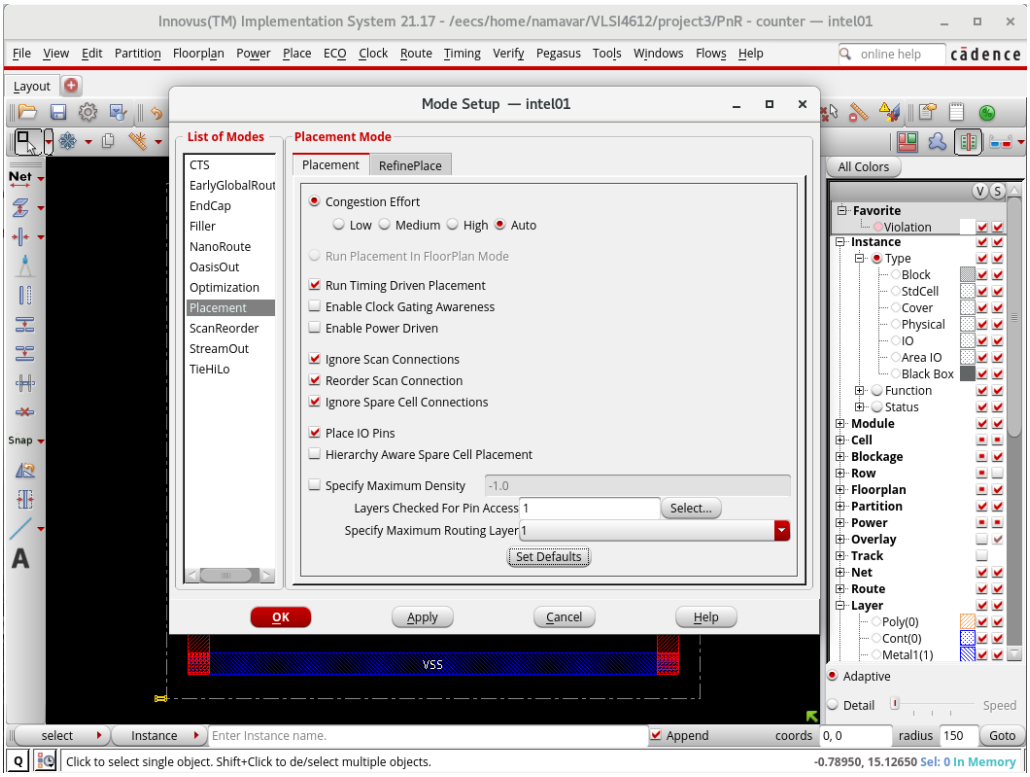

Figure 17: Placement Mode window. Click **Set Defaults** before proceeding.

Once placement is complete, the standard cells will be positioned within the core area, as shown in Figure 18.

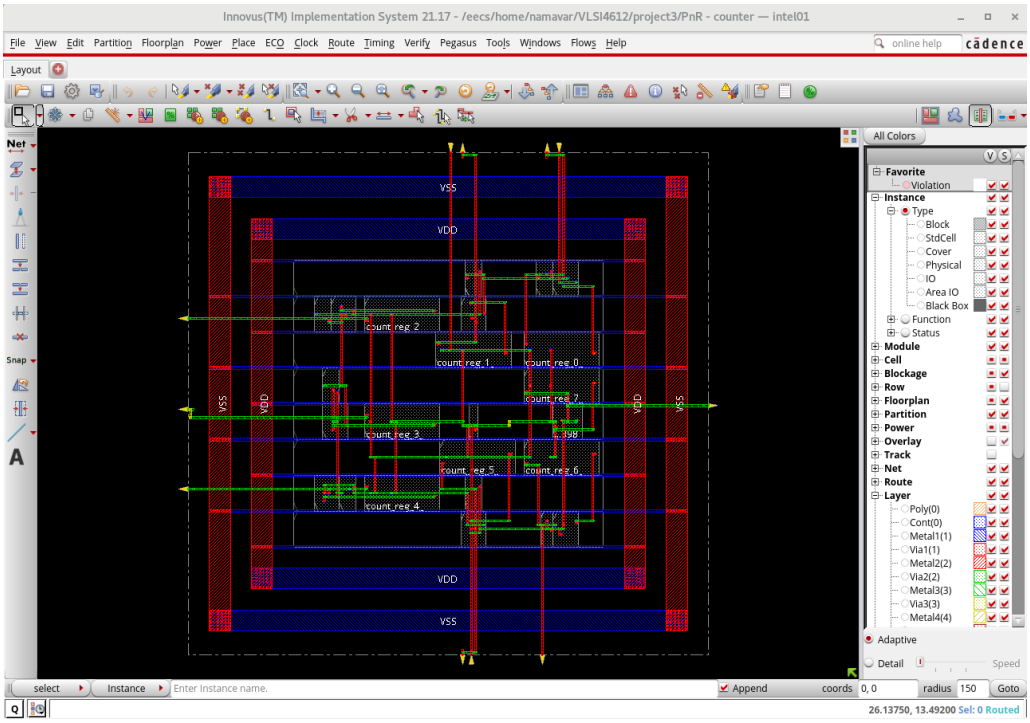

Figure 18: Standard cell placement after running the placement tool.

5.8 Clock Tree Synthesis (CTS)

In a digital circuit, the *clock tree* is a critical component responsible for distributing the clock signal to all sequential elements, such as flip-flops and registers. The goal of **Clock Tree Synthesis (CTS)** is to minimize **clock skew** (the difference in arrival time of the clock signal at different registers) and **clock transition time**, while ensuring that the clock signal meets all timing constraints.

Without an optimized clock tree, clock signals may reach different parts of the circuit at different times, leading to setup and hold timing violations. CTS ensures that the clock network is **balanced, meets timing constraints**, and minimizes **power consumption**.

5.8.1 Executing Clock Tree Synthesis in Innovus

To synthesize the clock tree, execute the following commands in the **Innovus shell** (the terminal where you launched Innovus):

```
create_ccopt_clock_tree_spec
set_ccopt_property target_max_trans 0.5
set_ccopt_property target_skew 0.05
ccopt_design
```

These commands perform the following:

- `create_ccopt_clock_tree_spec`: Generates a **clock tree specification file** that defines the clock constraints.

- `set_ccopt_property target_max_trans 0.5`: Sets the **maximum transition time** (clock rise/fall time) to **0.5 ns**.
- `set_ccopt_property target_skew 0.05`: Sets the **target clock skew** (the maximum difference between clock arrival times) to **50 ps (0.05 ns)**.
- `ccopt_design`: Runs the **clock tree optimization (CCOpt)** process, balancing the clock network.

Once CTS is complete, Innovus will generate **clock tree reports**, which can be analyzed to verify that clock constraints (skew, transition, and latency) are met.

Note: Since your technology library (`.lib`) uses **nanoseconds (ns)** as the timing unit, the values for **maximum transition time and clock skew** have been set accordingly.

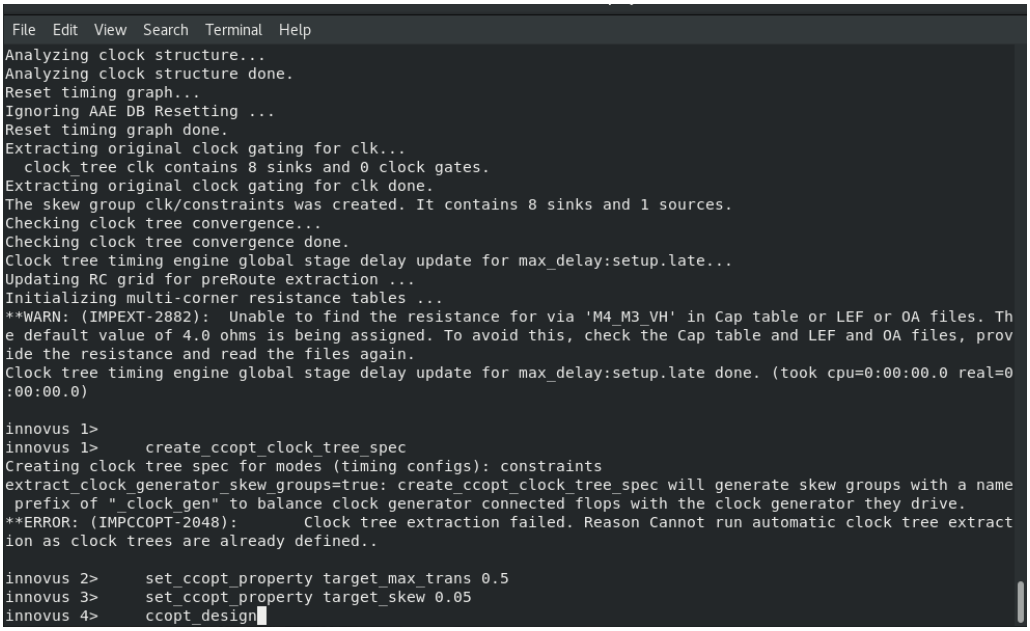

Figure 19: Clock Tree Synthesis execution in Innovus.

5.9 Final Routing with NanoRoute

The final step in place-and-route (PnR) is to complete the full routing process using **NanoRoute**. This ensures that all standard cells and macros are properly connected while maintaining design rule constraints.

To perform the final routing, follow these steps:

1. Click **Route** → **NanoRoute** → **Route**.
2. The **NanoRoute** window will appear, as shown in Figure 20.
3. Click **OK** to start the routing process.

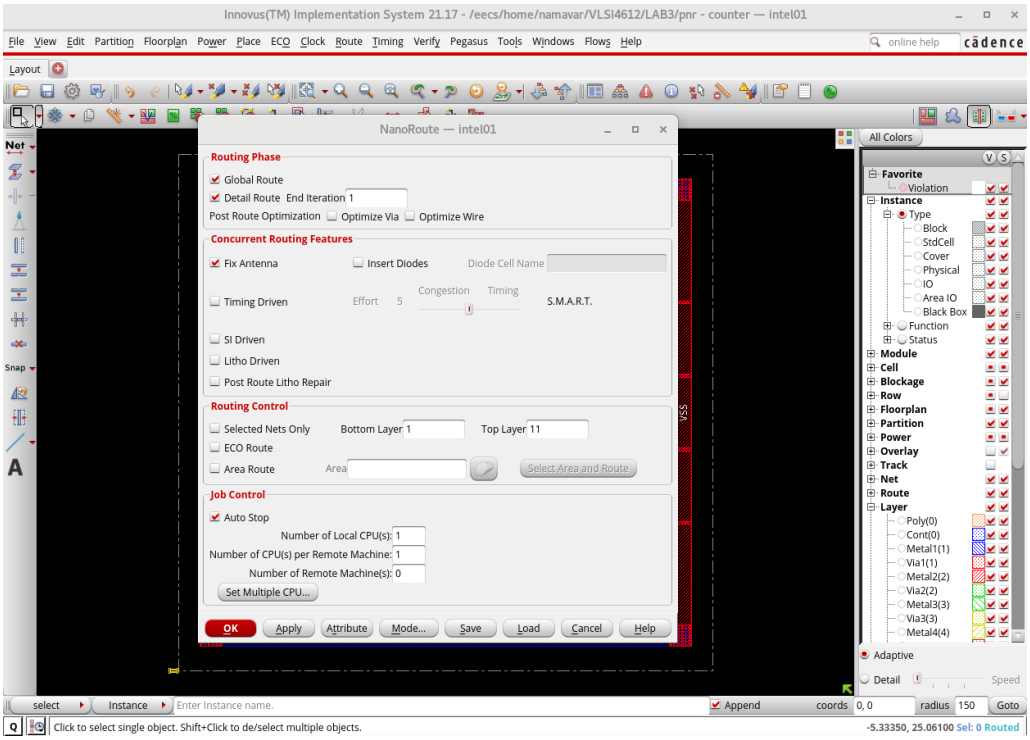

Figure 20: NanoRoute window for performing final routing.

Once the routing is completed, you will see the final layout with all interconnections routed, as shown in Figure 21.

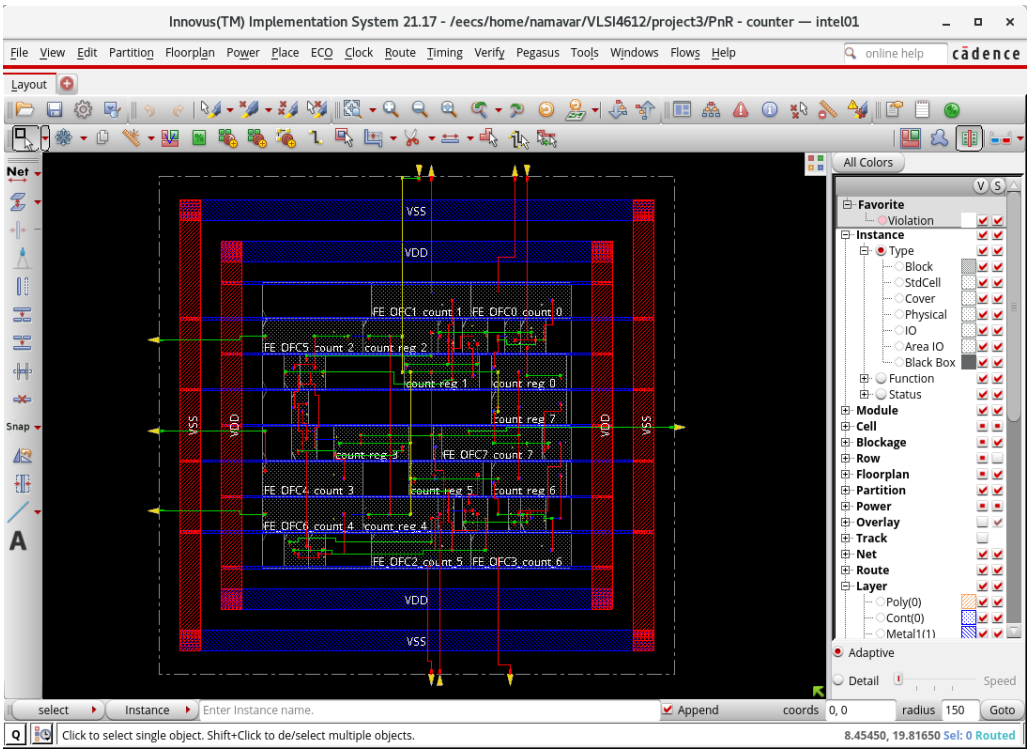

Figure 21: Final routing results after running NanoRoute.

5.10 Filler Placement

After completing the routing process, gaps may remain between standard cells in the core area. These gaps can lead to **Design Rule Check (DRC)** violations because they may break the continuity of power (VDD) and ground (VSS) connections. To prevent this, **filler cells** are added to fill these gaps, ensuring proper connectivity and avoiding DRC violations.

To add filler cells in Innovus, follow these steps:

1. Click **Place** → **Physical Cell** → **Add Filler**.
2. The **Add Filler** window will appear, as shown in Figure 22.
3. Click on **Select** to choose filler cells.
4. In the **Select Filler Cells** window, add all **FILLX** cells to the **Selectable Cell List**.
5. Click **Close**, then click **OK** in the **Add Filler** window.

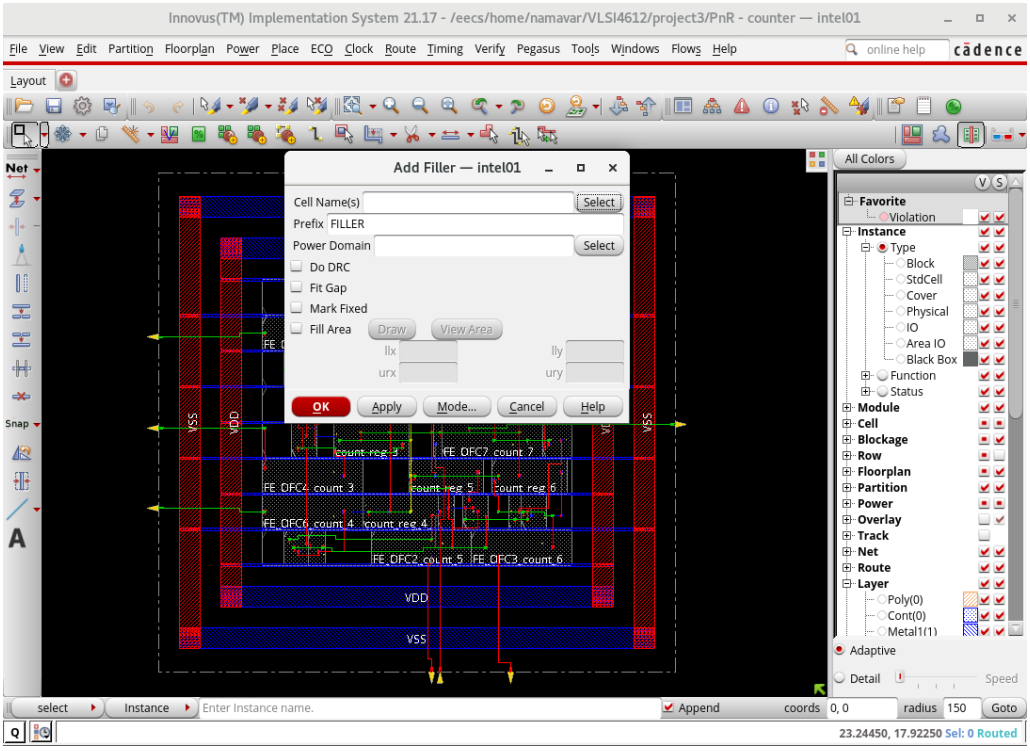

Figure 22: Add Filler window in Innovus.

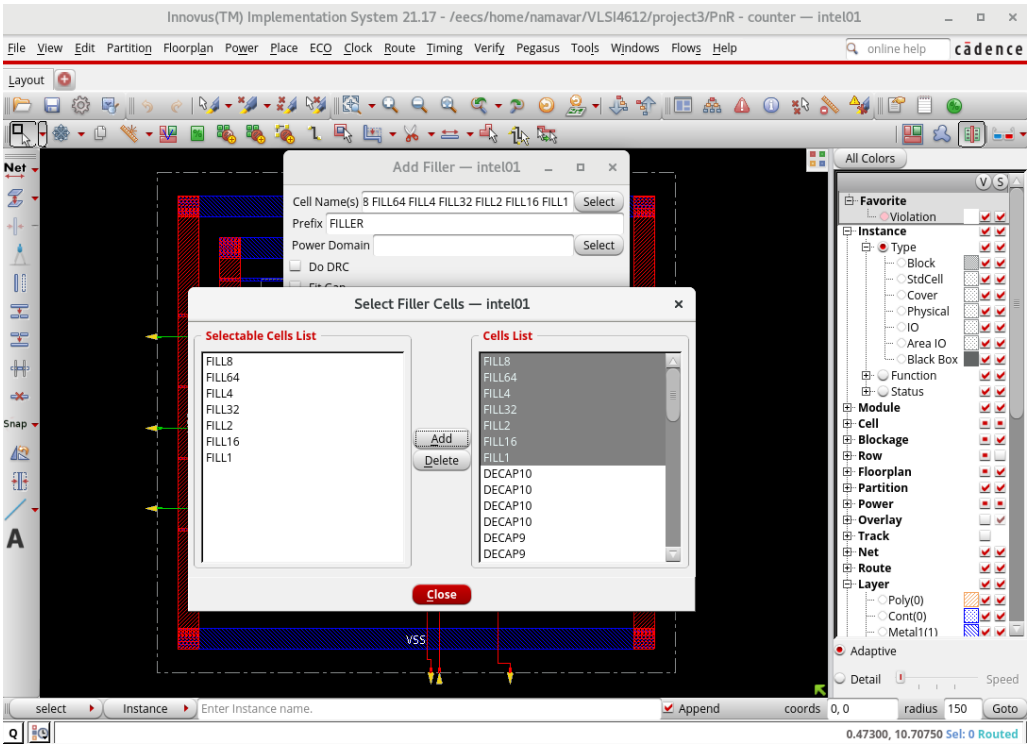

Figure 23: Selecting filler cells for placement.

Once the filler cells are placed, the final result should resemble Figure 24. As seen in the figure, the empty spaces in the core area are now filled with filler cells, ensuring continuous power and ground connections.

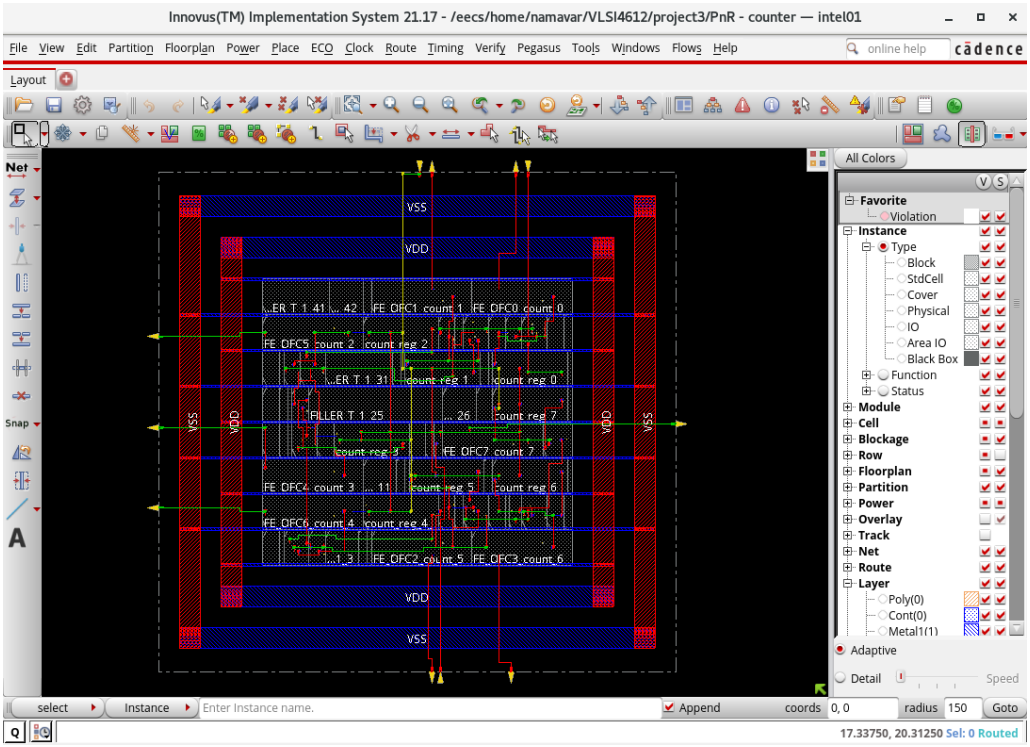

Figure 24: Final layout after filler placement.

5.11 Verify DRC and Connectivity

After completing the routing step, it is essential to verify that the layout meets the **Design Rule Check (DRC)** constraints and ensure proper **connectivity of all nets** in the design. These checks help identify and correct potential violations before finalizing the design.

5.11.1 Verifying DRC

To perform **Design Rule Check (DRC)** in **Innovus**, follow these steps:

1. Click on **Verify** → **Verify DRC...**
2. The **Verify DRC** window will appear, as shown in Figure 25.
3. Go to the **Advanced** tab.
4. Check the option **Excludes The Checking of PG Net** to avoid unnecessary warnings.
5. Click **OK** to run the DRC verification.

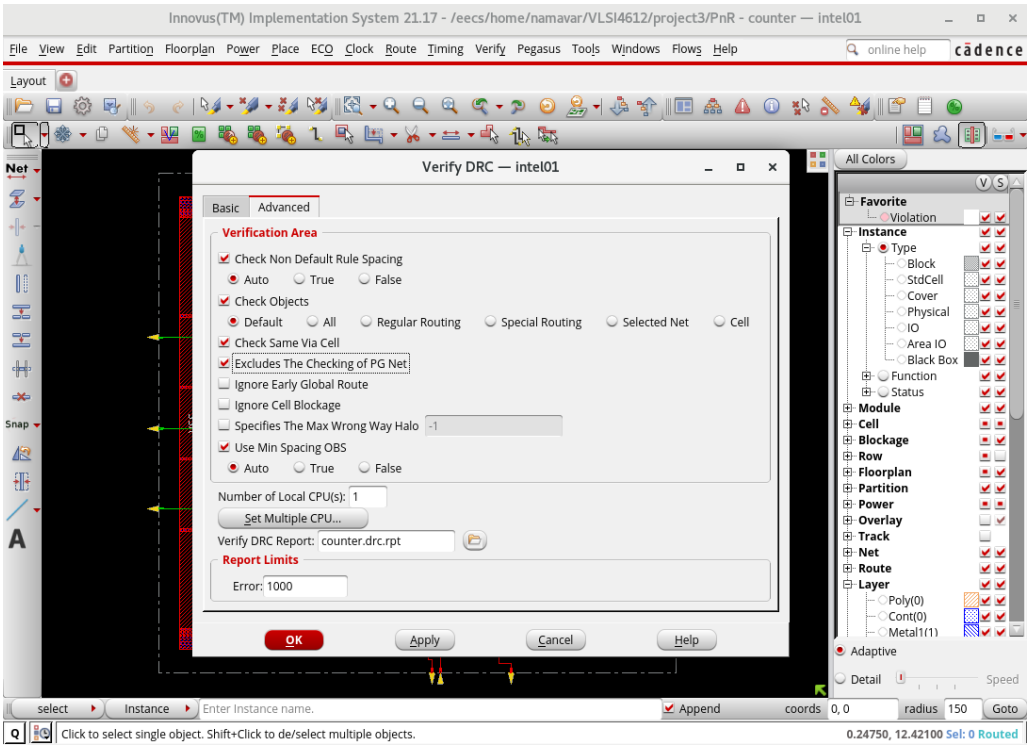

Figure 25: Verify DRC window in Innovus.

Once the DRC verification is complete, you will see the results in the **Innovus shell window**, as shown in Figure 26.

```
File  Edit  View  Search  Terminal  Help
#Default setup view is reset to worst_case.
AAE_INFO: Post Route call back at the end of routeDesign
#routeDesign: cpu time = 00:00:01, elapsed time = 00:00:01, memory = 1982.89 (MB), peak = 2119.96 (MB)

*** Summary of all messages that are not suppressed in this session:
Severity  ID               Count  Summary
WARNING   NRDB-2005            4  %s %s has special wires but no definitio...
WARNING   NRIG-1303            2  The congestion map does not match the GC...
WARNING   NRIF-90              1  Option setNanoRouteMode -routeBottomRout...
WARNING   NRIF-91              1  Option setNanoRouteMode -routeTopRouting...
*** Message Summary: 5 warning(s), 0 error(s)

innovus 5> #-check_ndr_spacing auto               # enums={true false auto}, default=auto,  user setting
#-check_same_via_cell true              # bool, default=false, user setting
#-exclude_pg_net true                   # bool, default=false, user setting
#-report counter.drc.rpt                # string, default="", user setting
 *** Starting Verify DRC (MEM: 2557.1) ***

  VERIFY DRC ..... Starting Verification
  VERIFY DRC ..... Initializing
  VERIFY DRC ..... Deleting Existing Violations
  VERIFY DRC ..... Creating Sub-Areas
  VERIFY DRC ..... Using new threading
  VERIFY DRC ..... Sub-Area: {0.000 0.000 24.800 23.940} 1 of 1
  VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

  Verification Complete : 0 Viols.

 *** End Verify DRC (CPU: 0:00:00.0  ELAPSED TIME: 0.00  MEM: 256.1M) ***

innovus 5>
```

Figure 26: DRC verification results in the Innovus shell.

5.11.2 Verifying Connectivity

To ensure all nets are properly connected, perform the **Connectivity Verification**:

1. Click on **Verify** → **Verify Connectivity...**
2. The **Verify Connectivity** window will appear, as shown in Figure 27.
3. Click **OK** to run the connectivity verification.

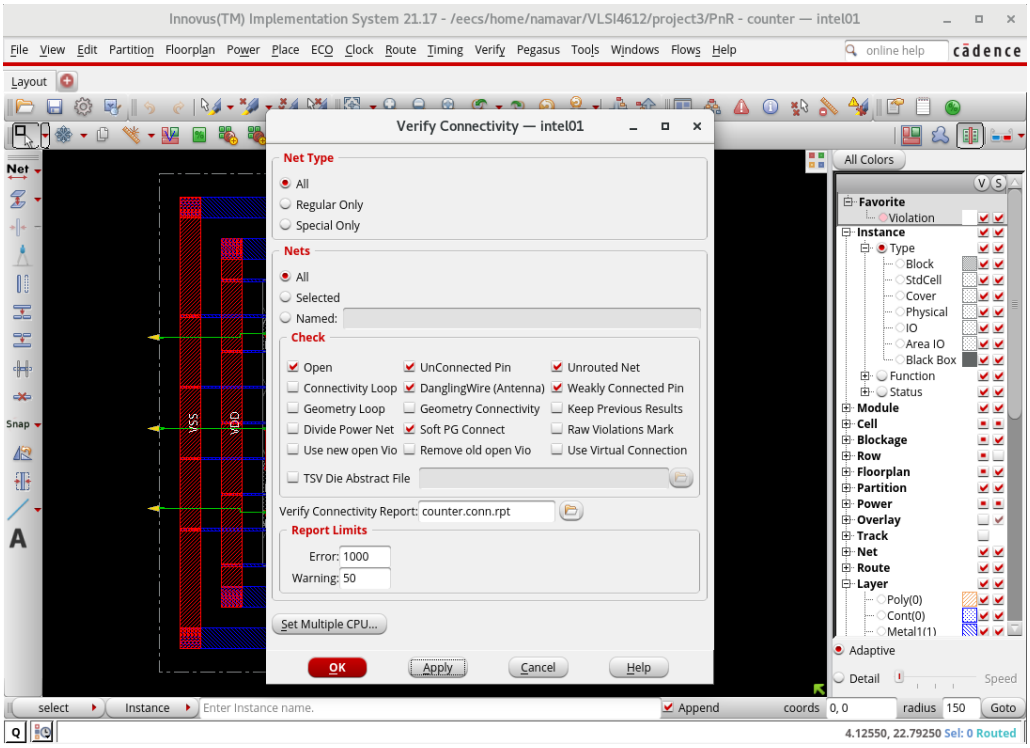

Figure 27: Verify Connectivity window in Innovus.

After the verification completes, the results will be displayed in the **Innovus shell**, as shown in Figure 28.

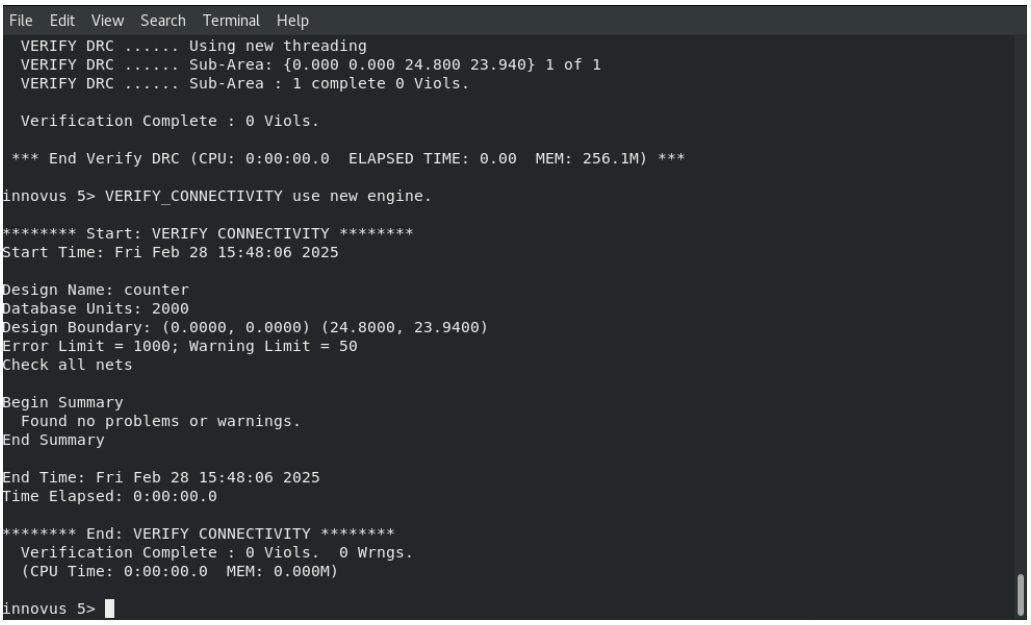

Figure 28: Connectivity verification results in the Innovus shell.

If no violations are detected, the **layout is considered valid**, and you can proceed to the **final GDSII export step**.

5.12 Exporting GDSII from Innovus

After completing the Place and Route (PnR) process, the final step before verification is to export the design in **GDSII** format. The GDS file contains the full layout information and is used for further analysis and verification in **Cadence Virtuoso**.

To export the design as a GDSII file, follow these steps:

1. Click **File** → **Save** → **GDS/OASIS...**.
2. The **GDS/OASIS Export** window will appear, as shown in Figure 29.
3. In the **Output File** field, specify the directory and name for the output GDS file.
4. In the **Map File** section, browse and select the appropriate map file as shown in Figure 30.
5. Ensure that the **Merge Files** option is checked. This step is essential to combine all design layers properly.
6. Verify that the correct GDS output file is selected.
7. Click **OK** to complete the export process.

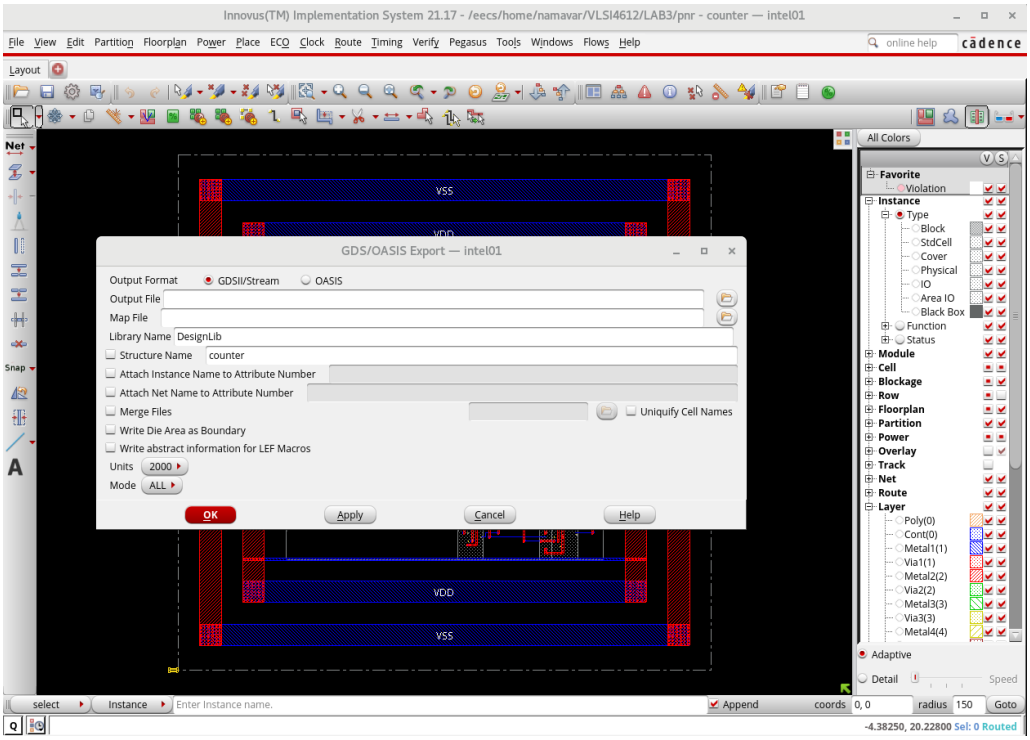

Figure 29: GDS/OASIS Export window in Innovus.

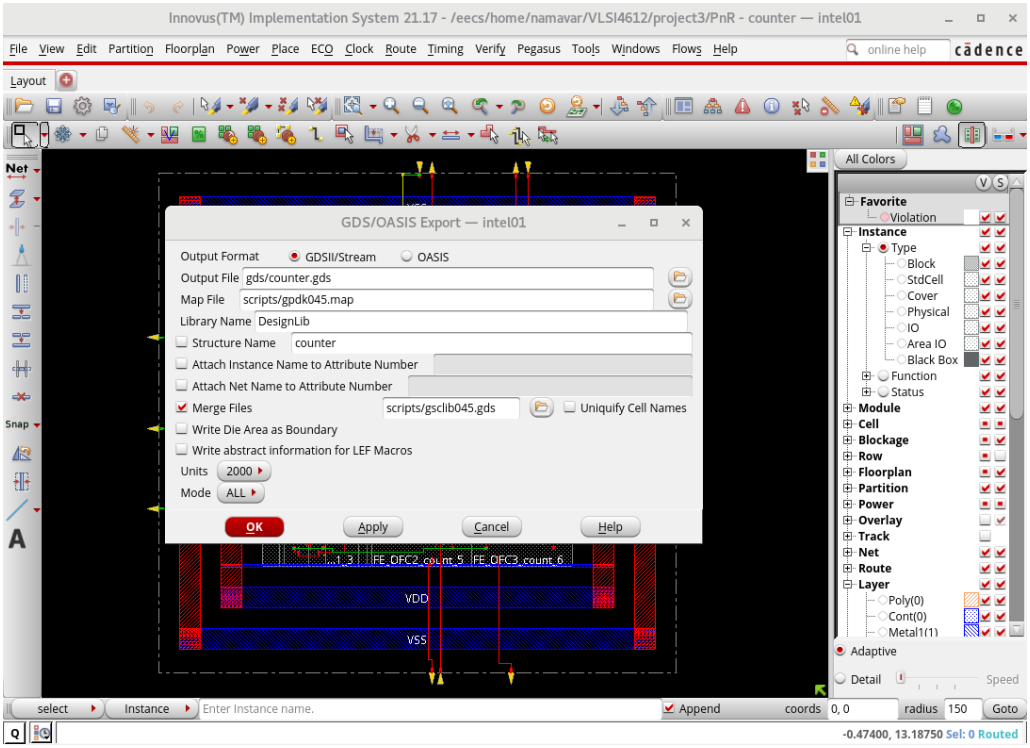

Figure 30: Specifying the output file, selecting the proper map file, and enabling the merge files option for GDS export.

If everything is set up correctly, the exported GDS file should be visible in the specified directory, as shown in Figure 31.

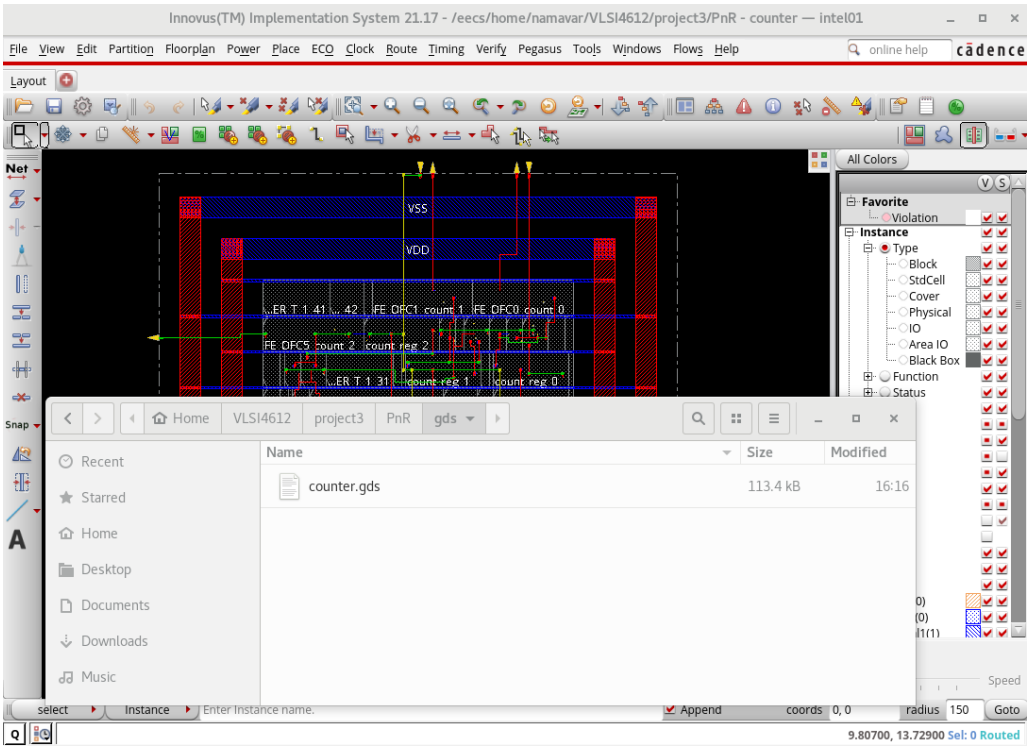

Figure 31: Generated GDSII file in the output directory.

5.13 Importing the Generated Layout into Virtuoso

After completing the Place and Route (PnR) process in Cadence Innovus, the next step is to import the generated GDSII layout into Cadence Virtuoso for further verification and final checks.

First, close Innovus and the corresponding terminal session.

5.13.1 Launching Virtuoso and Creating a Library

To launch Virtuoso, open a terminal in the `EECS4612` directory and run:

```
gpdk45
```

Once Virtuoso has launched, follow these steps:

1. Open the **Library Manager**.
2. Create a new library by clicking **File** → **New** → **Library**.
3. Name the library `Counter_PnR`.
4. Attach the library to the **gpdk45** technology file.

5.13.2 Importing the GDSII File

To import the generated GDSII layout:

1. In the Virtuoso main window, click **File** → **Import** → **Stream**.

2. The **XStream In** window will appear, as shown in Figure 32.
3. In the **Stream File** field, browse to select the GDSII file exported from Innovus.
4. In the **Library** field, select `Counter_PnR`.
5. Click **Translate**.

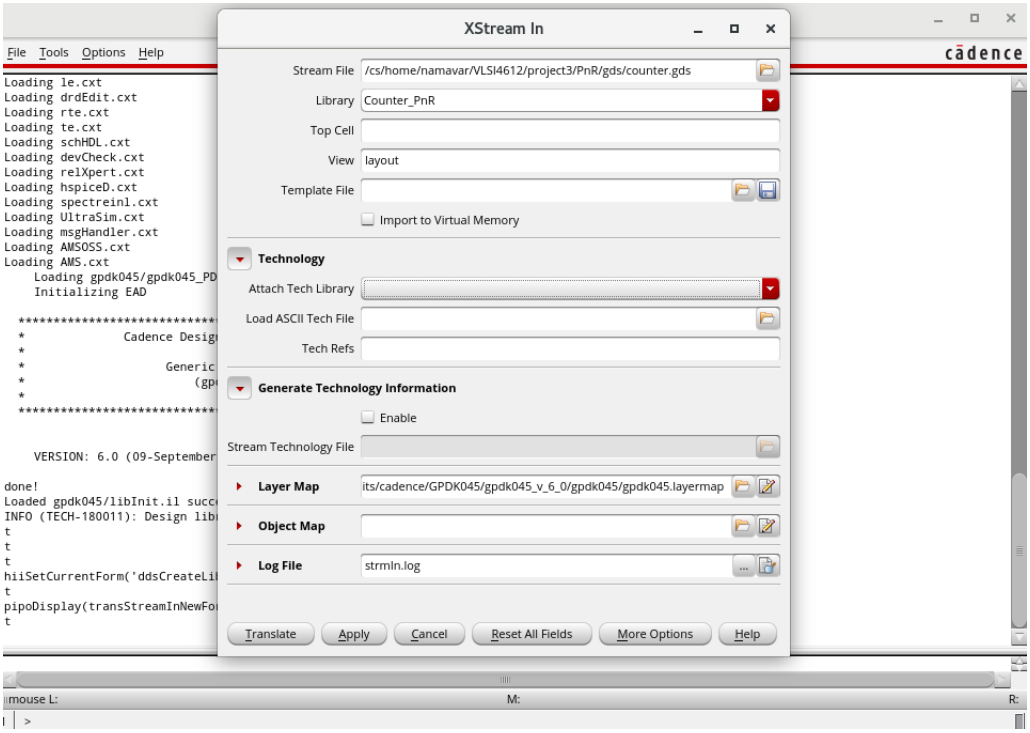

Figure 32: XStream In window for GDSII import in Virtuoso.

Once the translation is complete, a message will appear as shown in Figure 33. Click **Yes** to proceed and ignore any warnings.

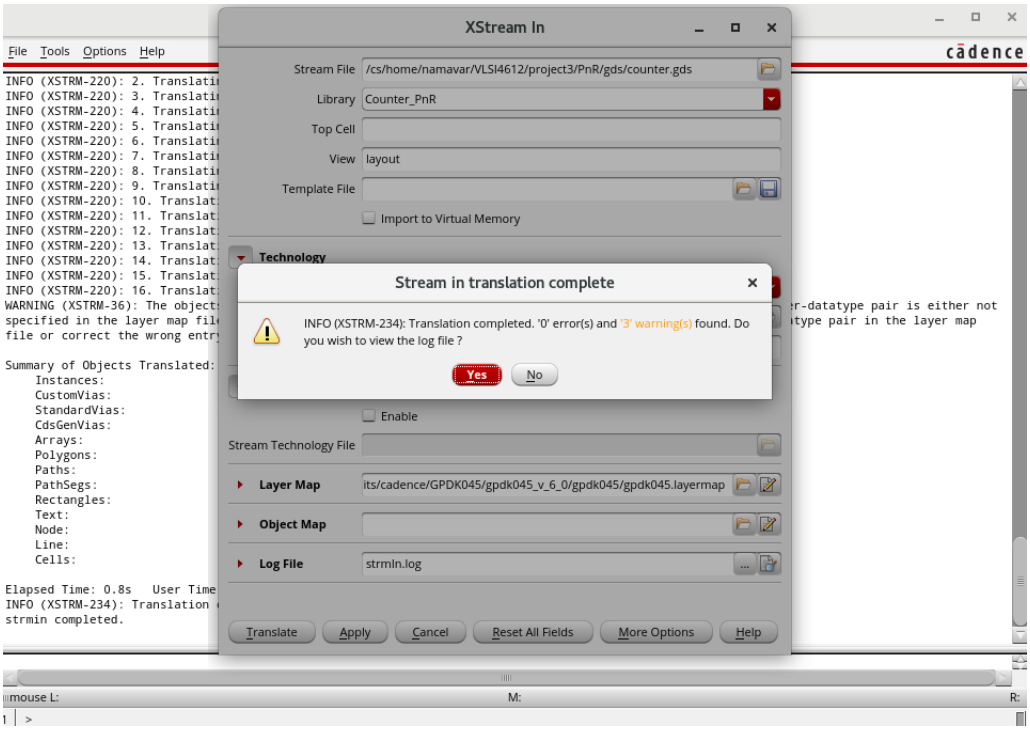

Figure 33: Translation completion message for Stream In process.

5.13.3 Opening the Layout in Virtuoso

To view the imported layout:

1. Open the **Library Manager**.
2. Navigate to the `Counter_PnR` library.
3. Locate the `counter` cell and open the `layout` view, as shown in Figure 34.

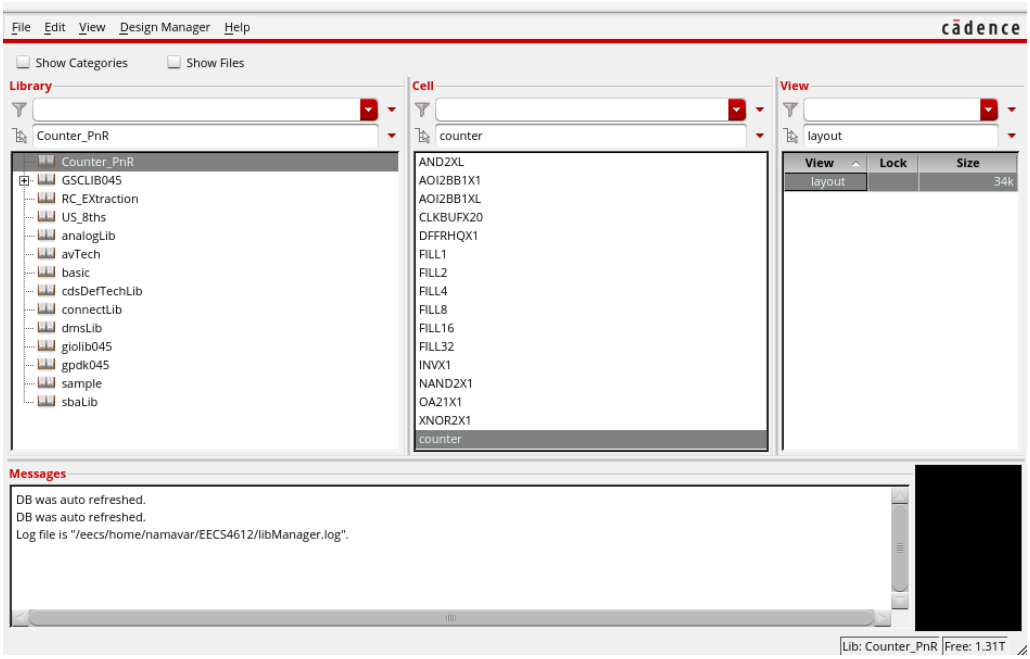

Figure 34: Library Manager displaying the imported layout cell.

If the layout appears but lacks detail, press **Shift + F** to display all layers, as shown in Figure 35.

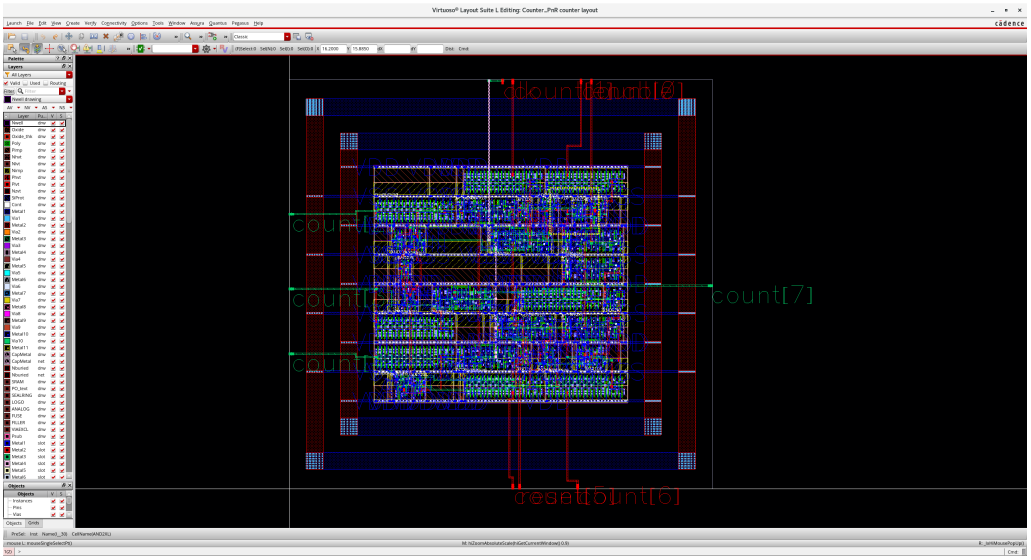

Figure 35: Counter layout displayed in Virtuoso after import.

5.14 Conclusion

In this section, we successfully completed the automatic layout generation process using Cadence Innovus and imported the final GDSII file into Cadence Virtuoso. The process included:

- Synthesizing the RTL design into a netlist.
- Performing Place and Route (PnR) to generate a physical layout.
- Running post-PnR verification steps such as DRC and connectivity checks.
- Exporting the GDSII file and importing it into Virtuoso for final visualization.

This workflow is a fundamental part of the digital design process, enabling students to transition from RTL coding to physical layout implementation. Further steps, such as final verification and signoff checks, would be necessary for a complete ASIC design flow.

Appendix 1 - Final Project Tips

1. Placement and Special Routing Order

In the first version of this tutorial, **Special Routing** was performed immediately after **Placement**. However, for better consistency, the order of **Placement** and **Special Routing** has been changed. Please make sure to follow the updated workflow as outlined in this tutorial.

2. Dummy Clock for ALU Design

The **Final Project** involves designing an **ALU (Arithmetic Logic Unit)**, which does not have a clock signal. However, a **dummy clock** is required for **timing analysis** in Innovus.

Before synthesizing your design, **edit the** `genus_script.tcl` **script** and modify the clock frequency to **1 MHz**:

```
set clockFrequency 1                    ;# Clock frequency in MHz
```

3. Listing HDL Files for Synthesis

Since the **ALU** consists of multiple **sub-modules**, make sure that **all Verilog HDL files** are listed correctly in the `genus_script.tcl` file. Failing to list a sub-module will cause errors during synthesis.

4. Pre-Synthesis Simulation Filelist

For **pre-synthesis simulation**, you need to list all **HDL files** in the `filelist.f` file. Modify this file carefully, ensuring that:

- Each **sub-module** is listed before the **top module**.
- The **Makefile** has the correct top module name.

If you want to simulate a **specific sub-module**, update the **Makefile** accordingly.

5. Top Module Name in Scripts

Ensure that the **top module name** is correctly set in:

- **Synthesis script** (`genus_script.tcl`)
- **Place and Route script** (`init.globals`)

6. Using the Pre-Designed Pad Frame

A **pre-designed Pad Frame** has been shared for your **Final Project**. It consists of:

- **71 Input Pads**
- **33 Output Pads**

- **4 VSS Pads**
- **4 VDD Pads**

These match the required **inputs and outputs** of a **32-bit ALU**. The Pad Frame is available on **eClass** as a ZIP file.

1. Download and extract `padFrame.zip` inside your home directory.
2. Navigate to the extracted folder (`ALU_DESIGN`) and run:

```
gpdk45
```

To assist you, **VDD/VSS Pads and Output Pads are labeled** to distinguish them from **Input Pads**, as shown in Figure 36.

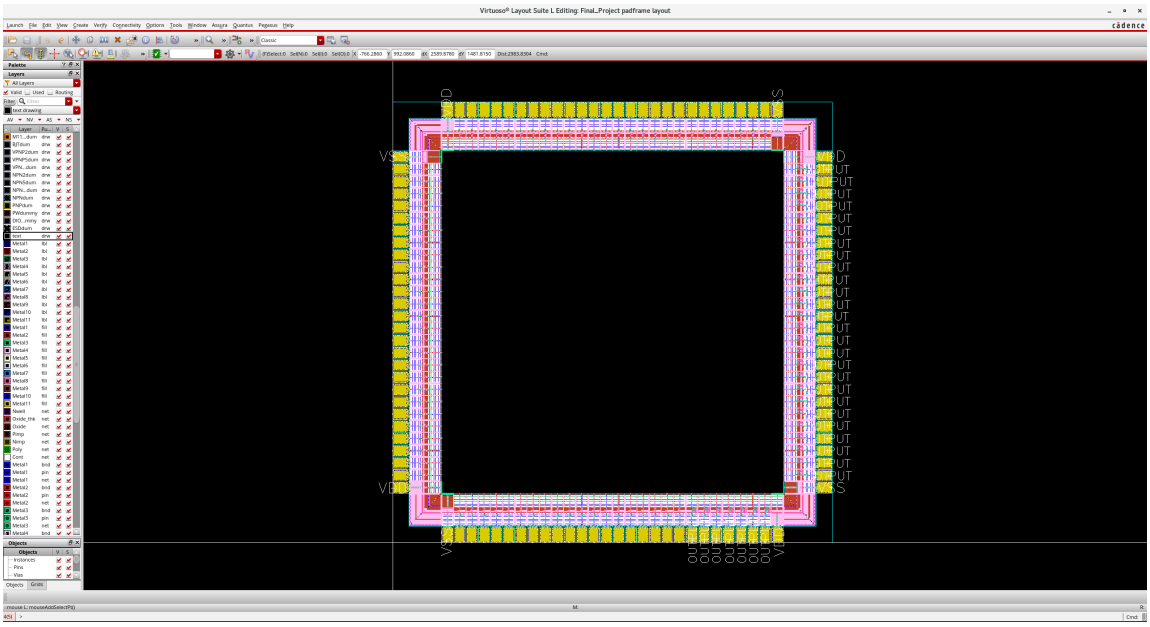

Figure 36: Pad Frame for the Final Project

Additionally, an example **Output Pad** is illustrated in Figure 37, demonstrating how outputs should be connected.

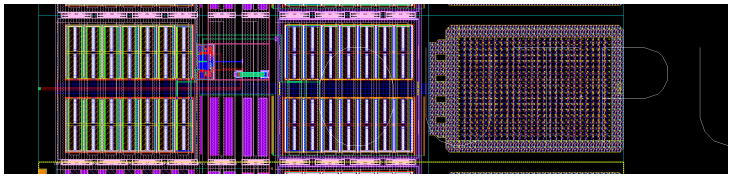

Figure 37: Sample Output Pad for ALU

Once the **ALU layout** is imported into Virtuoso, **place the ALU inside the Pad Frame** and **connect the ALU pins to the respective pads** using the appropriate **metal layers and vias**, as illustrated in Figure 38.

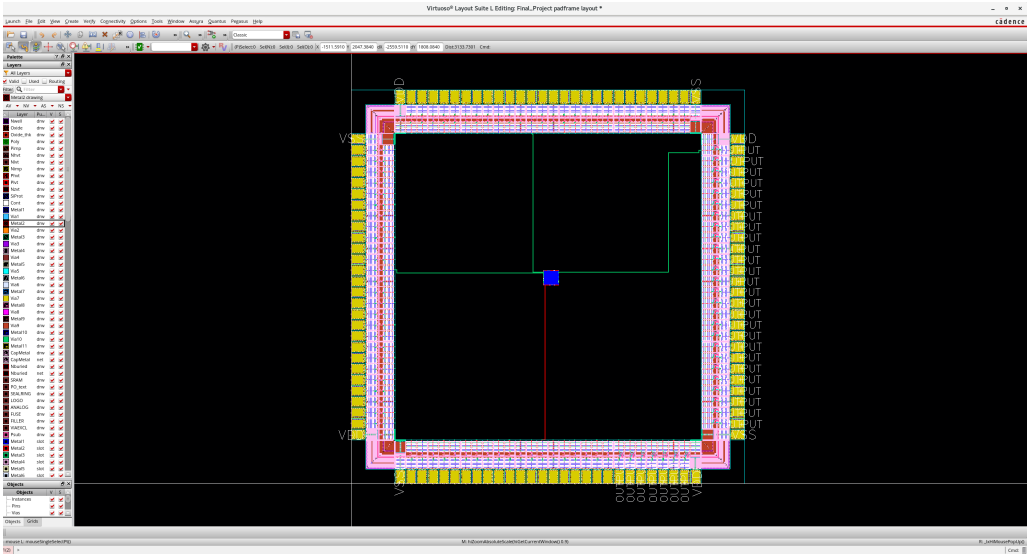

Figure 38: Example of ALU pin to pad connection using metal layers and vias.

7. I/O Assignment for Placement

To help integrate your final layout with the **Pad Frame**, an **I/O Assignment File** has been provided. This file ensures proper **pin order and locations** during **placement** and is automatically copied when you initialize your project (by executing the setup commands).

The file is located at:

```
./PnR/scripts
```

If you wish to use this file, you must modify the `init.globals` script by **uncommenting** the following line:

```
#set init_io_file {./scripts/pinAssignment.io}
```

This will allow Innovus to correctly load the I/O pin assignments during the placement stage. If you are interested in understanding more about **I/O assignment**, you are encouraged to open and explore the file.

Standardized Input and Output Port Names

If you decide to use the provided file, you must ensure the correct port naming. To prevent errors when loading your design into Innovus, it is **crucial** to use the following standardized **input and output port names** in your **32-bit ALU Verilog file**:

Inputs:

```
A[31:0], B[31:0], S[3:0], CIN, DR, DL
```

Outputs:

```
F[31:0], COUT
```

Appendix 2 - Timing Analysis in Innovus

Since this project does not have a real clock port, you do not need to perform **Clock Tree Synthesis (CTS)**. However, you still need to analyze the timing of the circuit after the final routing stage.

Generating the Timing Report

To save the timing reports, run the following commands in the **Innovus shell** after completing the **final routing** step:

```
setAnalysisMode -analysisType onChipVariation

timeDesign -postRoute -pathReports -drvReports -slackReports \
-numPaths 100 -prefix postRoute -outDir timingReports
```

After executing this command, navigate to the `timingReports` directory, extract the `postRoute_all.tarpt.gz` file, and open the extracted file.

Step 1: Understanding the Key Terms in the Report

The extracted timing report provides essential details about the circuit's timing performance. An example of a post-route timing report is shown in Figure 39.

```
Path 1: MET Late External Delay Assertion
Endpoint:   F[31]     (v) checked with  leading edge of 'clk'
Beginpoint: Select[2] (v) triggered by  leading edge of 'clk'
Path Groups: {clk}
Analysis View: worst_case
Other End Arrival Time          0.000
- External Delay               0.030
+ Phase Shift                1000.000
= Required Time              999.970
- Arrival Time                61.662
= Slack Time                 938.308
     Clock Rise Edge                     0.000
     + Input Delay                       0.010
     + Drive Adjustment                  0.008
     = Beginpoint Arrival Time           0.018
     +-----------------------------------------------------------------------------------------------------------------------+
     |                  Pin                  | Edge |              Net               |   Cell   | Delay | Arrival | Required |
     |                                       |      |                                |          |       |  Time   |   Time   |
     |---------------------------------------+------+--------------------------------+----------+-------+---------+----------|
     | Select[2]                             |  v   | Select[2]                      |          |       |   0.018 |  938.326 |
     | g2334__5107/A                         |  v   | Select[2]                      | OR2X1    | 0.000 |   0.018 |  938.326 |
     | g2334__5107/Y                         |  v   | n_294                          | OR2X1    | 0.134 |   0.152 |  938.460 |
     | g2333__2398/AN                        |  v   | n_294                          | NAND2BX1 | 0.000 |   0.152 |  938.460 |
     | g2333__2398/Y                         |  v   | n_293                          | NAND2BX1 | 0.142 |   0.294 |  938.602 |
     | g2/AN                                 |  v   | n_293                          | NAND2BX1 | 0.000 |   0.294 |  938.602 |
     | g2/Y                                  |  v   | n_1254                         | NAND2BX1 | 0.142 |   0.436 |  938.744 |
     | inc_add_53_27_Y_dec_sub_31_27_drc_bufs817/A |  v   | n_1254                         | INVX2    | 0.000 |   0.436 |  938.744 |
     | inc_add_53_27_Y_dec_sub_31_27_drc_bufs817/Y |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_1 | INVX2    | 0.337 |   0.773 |  939.081 |
     | inc_add_53_27_Y_dec_sub_31_27_g810__2346/CI |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_1 | ADDFX1   | 0.001 |   0.774 |  939.082 |
     | inc_add_53_27_Y_dec_sub_31_27_g810__2346/CO |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_3 | ADDFX1   | 0.441 |   1.215 |  939.523 |
     | inc_add_53_27_Y_dec_sub_31_27_g809__2883/CI |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_3 | ADDFX1   | 0.000 |   1.215 |  939.523 |
     | inc_add_53_27_Y_dec_sub_31_27_g809__2883/CO |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_5 | ADDFX1   | 0.196 |   1.411 |  939.719 |
     | inc_add_53_27_Y_dec_sub_31_27_g808__9945/CI |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_5 | ADDFX1   | 0.000 |   1.411 |  939.719 |
     | inc_add_53_27_Y_dec_sub_31_27_g808__9945/CO |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_7 | ADDFX1   | 0.201 |   1.612 |  939.921 |
     | inc_add_53_27_Y_dec_sub_31_27_g807__9315/CI |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_7 | ADDFX1   | 0.000 |   1.612 |  939.921 |
     | inc_add_53_27_Y_dec_sub_31_27_g807__9315/CO |  ^   | inc_add_53_27_Y_dec_sub_31_27_n_9 | ADDFX1   | 0.210 |   1.822 |  940.130 |
```

Figure 39: Example of a Post-Route Timing Report from Innovus

The key terms to understand in this report are:

- **Required Time:** This is the latest possible time by which the signal must arrive at the capturing element. In the report, the required time is given as **999.970 ns**, which corresponds to the clock period (set to **1000 ns**, adjusted for external delays and phase shifts).

- **Arrival Time:** This represents the cumulative delay experienced along a particular path, from the launching point to the endpoint. For instance, one path (Path 1) may show an arrival time of **61.662 ns**.
- **Slack:** Slack is calculated using the formula:

$$\text{Slack} = \text{Required Time} - \text{Arrival Time} \tag{1}$$

  A **positive slack** means that the path meets the timing requirements, whereas a **negative slack** indicates a timing violation.
- **Critical Path:** The **critical path** is the path with the **longest delay** (i.e., the highest arrival time) or equivalently, the path with the **smallest slack**.

  In the report, if the path ending at `F[31]` has the highest arrival time (e.g., **61.662 ns**), then it is the critical path.

Step 2: Identifying the Maximum Delay

To find the maximum delay:

1. Open the extracted timing report.
2. Look at the **Arrival Time** values for each reported path.
3. The highest arrival time corresponds to the **critical path delay**.

For example, if the report shows the following values:

```
F[31] = 61.662 ns
F[30] = 61.347 ns
F[29] = 61.112 ns
```

Then, the **maximum delay** (critical path delay) is:

$$\text{Critical Path Delay} = 61.662 \text{ ns} \tag{2}$$

Step 3: Calculating the Maximum Operable Frequency

The **maximum clock frequency** ($f_{\max}$) at which the circuit can operate is given by the reciprocal of the **critical path delay**:

$$f_{\max} = \frac{1}{\text{Critical Path Delay}} \tag{3}$$

Using the previously found critical delay:

$$f_{\max} = \frac{1}{61.662 \times 10^{-9} \text{ s}} \approx 16.22 \text{ MHz} \tag{4}$$

Note: In real-world designs, additional margins are considered for **setup times, hold times, clock skew, and process variations**. However, for this basic calculation, the reciprocal of the maximum delay provides a reasonable estimate.

Appendix 3: Exporting the Final Layout from Virtuoso

Once your **design with the pad frame** is complete, you must **export your final GDSII file** for submission and further verification. Follow these steps:

1. In the **Virtuoso** main window, go to:

   `File` → `Export` → `Stream`

2. The **XStream Out** window will pop up, as shown in Figure 40.

3. Set the following parameters:

   - **Stream File:** Enter a proper name and path for the output GDS file.
   - **Library:** Select the **library** that contains your pad frame and design.
   - **Top Cell(s):** Choose the **pad frame cell view** that includes your design.
   - **Technology Library:** Set it to `gpdk045`.
   - **Layer Map:** Ensure the correct **layer mapping file** is set.

4. Click **Translate** to export the final GDSII file.

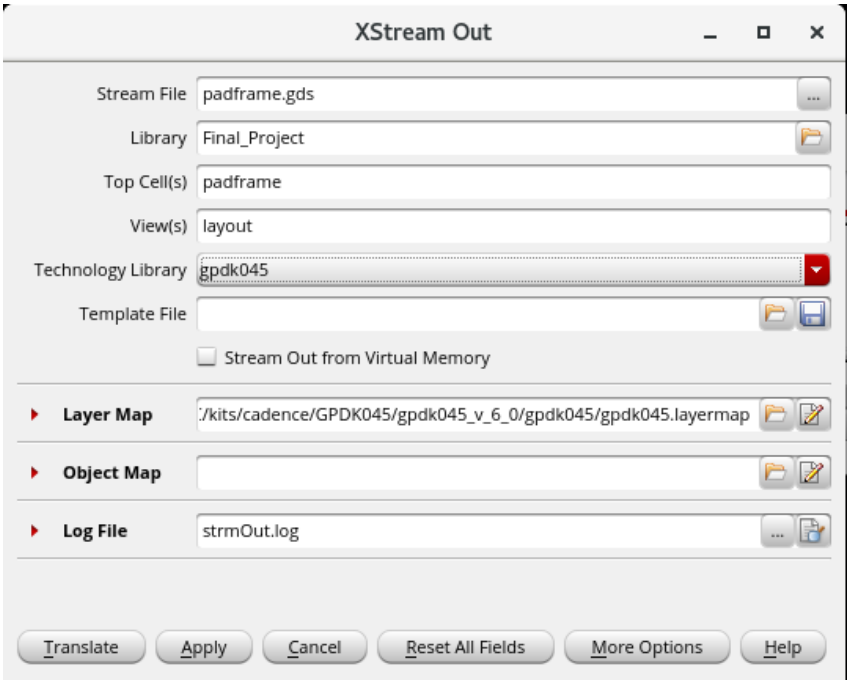

Figure 40: Exporting the Final GDSII File from Virtuoso

After completing this step, your **final layout** will be ready for submission and tape-out verification.