DOCUMENTATION OF PV

Date: 13-Feb-2025

Prepared by: Anik Balo

Table of Contents

Physical Verification (PV)

Physical Verification (PV) is a critical step in the VLSI design process. It ensures that the integrated circuit (IC) layout adheres to the manufacturing constraints and specifications. This step is essential to guarantee that the design can be reliably fabricated and will function as intended in real-world applications. After routing, your PnR tool should give you zero DRC/LVS violations. However, the PnR tool deals with abstracts like FRAM or LEF views. We use dedicated physical verification tools for signoff LVS and DRC checks.

PV signoff tools:

- **Calibre** - From Siemens
- **IC Validator** – From Synopsys
- **PVS** – From Cadence

PV Key Objective:

- **Design Rule Check (DRC)**: Ensures that the layout complies with the foundry's design rules, which are critical for manufacturability.
- **Layout Versus Schematic (LVS)**: Verifies that the physical layout matches the logical design represented by the schematic or netlist.
- **Electrical Rule Check (ERC)**: Identifies electrical issues such as shorts, opens, and other connectivity problems.
- **Antenna Check**: Detects and mitigates antenna effects that can occur during the fabrication process, potentially damaging the circuit.

LVS

Layout Versus Schematic (LVS) verification is a critical step in the VLSI design process. It ensures that the physical layout of the integrated circuit (IC) accurately represents the intended logical design as described by the schematic or netlist.

- LVS (Layout Versus Schematic) is a formal verification process. In essence, it operates on a binary principle: the two netlists are either entirely equivalent or they are not. There is no intermediate state, which can complicate the debugging process.
- The primary reference points are the top-level ports. The comparison process is explicitly instructed that port X in the source corresponds to port X in the layout. This serves as the 'ground truth,' allowing the tool to begin searching for equivalencies from these ports.
- If the design is found to be non-equivalent, the tool attempts to provide an informed guess. Unfortunately, this guess is not always highly accurate. However, it typically offers several hints that can guide us towards identifying fundamental issues

Input files:

- **Schematic Inputs:**
  - Netlist (SPICE, CDL or Verilog format) – Extracted from schematics
- **Layout Inputs:**
  - GDS files – The final layout database containing physical design data.
- **Rule Files:**
  - LVS Rule Deck – Foundry-provided file for the LVS tool
  - Device Recognition Rules – Defines how devices (transistors, resistors, capacitors) are recognized in the layout.
- **Other Supporting Files:**
  - SPICE Model Files – Used for transistor-level simulations and netlist comparison.
  - Configuration Files – Specifies tool settings (e.g., ignored mismatches, hierarchical comparisons).

In Physical Design LVS (Layout vs. Schematic), there are two types of netlists:

1. **Source Netlist (Schematic Netlist)** – Extracted from the schematic (logical design).
2. **Layout Netlist (Extracted Netlist)** – Extracted from the layout (physical design).

Source Netlist (Schematic Netlist):

- Represents the **ideal connectivity** from the circuit schematic.
- Typically in **CDL (Circuit Description Language)** or **SPICE (.sp) format**.
- Used as a reference in LVS comparison.

Format of Source Netlist (CDL/SPICE):

```
.SUBCKT NAND2 A B Y VDD VSS
M1 Y A VDD VDD PMOS L=0.1u W=0.5u
M2 Y B VDD VDD PMOS L=0.1u W=0.5u
M3 Y A N1 VSS NMOS L=0.1u W=0.3u
M4 N1 B VSS VSS NMOS L=0.1u W=0.3u
.ENDS NAND2
```

- .SUBCKT defines a logic cell.
- M1–M4 define transistors with their connections.
- Generated from schematic capture tools like Cadence Virtuoso or Synopsys Custom Compiler.

Layout Netlist (Extracted Netlist):

- Generated from **layout extraction** (post-layout netlist).
- Extracted from **GDSII/OASIS** using tools like **Calibre, IC Validator, or PVS**.
- May have additional parasitics (if RC extraction is included).

Format of Layout Netlist (Extracted CDL/SPICE):

```
.SUBCKT NAND2 A B Y VDD VSS
M1 Y A VDD VDD PMOS L=0.1u W=0.48u
M2 Y B VDD VDD PMOS L=0.1u W=0.52u
M3 Y A N1 VSS NMOS L=0.1u W=0.29u
M4 N1 B VSS VSS NMOS L=0.1u W=0.31u
.ENDS NAND2
```

- Same structure as the source netlist, but extracted from the layout.
- May have slight variations due to layout-dependent effects (LDE) or parasitics.

LVS flow:

- Preparation of the Source netlist
- Extraction of the Layout netlist
- Check for shorts, opens, and verify bulk connections (ERC)
- Comparison of Source vs. Layout

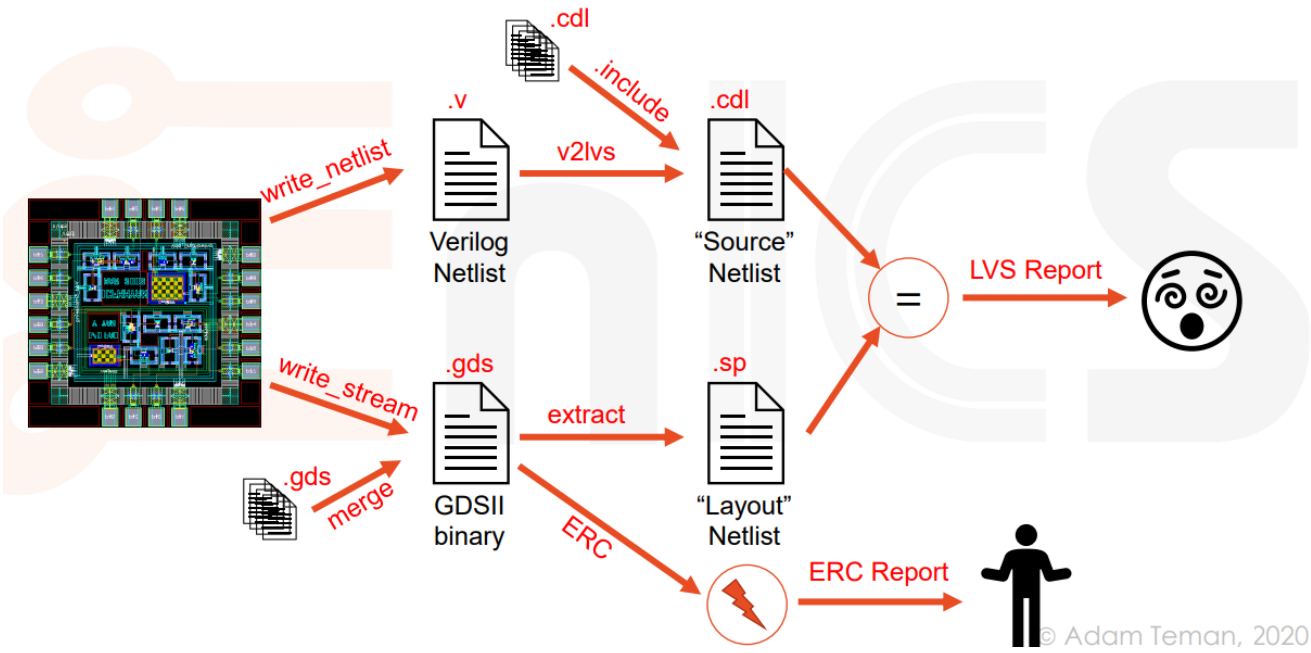


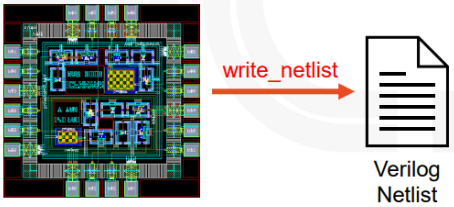
Fig: The complete Digital-on-top LVS flow [ref: Full chip DRC LVS lecture by Adam Teman]

• Write out Verilog netlist from Innovus

    - write_netlist -phys -exclude_leaf_cells -flatten_bus my_module.v
- Run v2lvs to create the "Source" SPICE netlist
    - v2lvs -sn -v my_verilog.v -o my_output_cdl.cdl -s my_includes_file.sp
- Write out GDSII from Innovus
    - write_stream my_layout.gds -merge $ALL_GDS -map_file $mapfile -unit 1000
- Extract "Layout" SPICE netlist from GDSII
    - calibre -hier -64 -hyper -turbo <num> -spice my_layout_netlist.sp runset.extract
- Compare "Source" and "Layout" Netlists
    - calibre -hier -64 -turbo -hcell heclls.txt runset.compare

Breaking LVS steps:

#Writing out the Verilog Netlist:



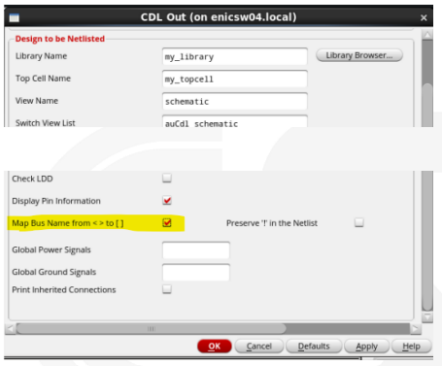
- **Basic command:**

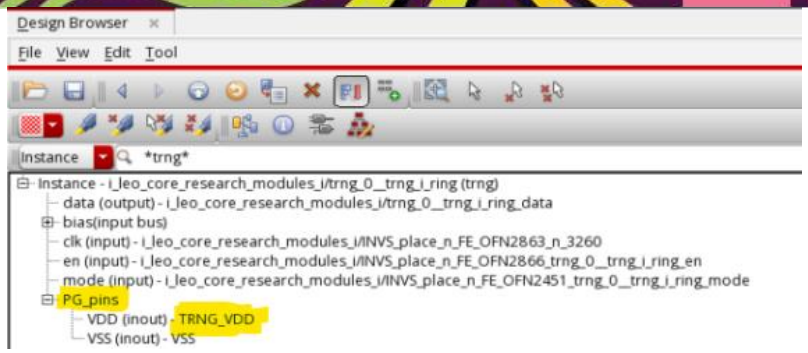
  write_netlist -phys -exclude_leaf_cells my_module.v

- **Frequent issues in this step:**
    - Global Net connectivity
    - Bus Notation
    - Assigns
    - Flipped Busses
    - Excluded Instances
    - Excluded Hierarchical Blocks

Problem #1: Missing Global Nets:

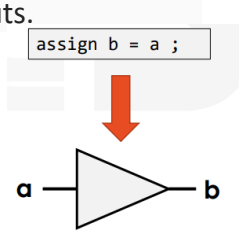
- Issue:
    - Logical connectivity (GTL netlist) doesn't require power nets
    - CMOS gates assume the existence of a logic '1' and a logic '0'
    - But these logic levels may come from different voltage sources
- Solution:
    - The write_netlist –phys flag writes out global power nets
    - However, you first need to initialize these in CPF/UPF or init_design:
        - set_db init_ground_nets
        - set_db init_power_nets
    - And you need to connect them to the right pins:
        - connect_global_net
    - To verify the connections, use the design browser:





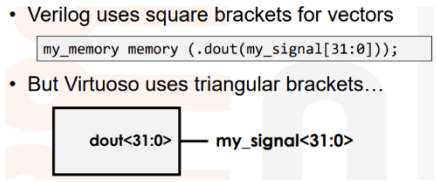
Problem #2: Assigns in your Netlist

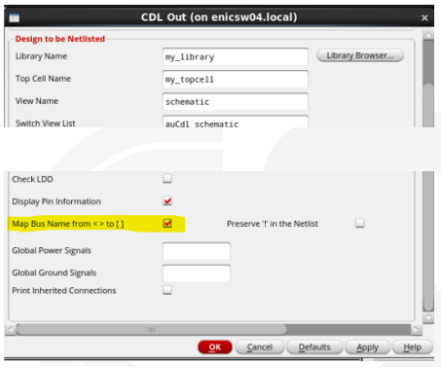
- **Issues:**
  - **Frequent Use of Assign Statements**: RTL code often utilizes the assign keyword in Verilog.
  - **Gate-Level Tool Compatibility**: Many gate-level tools may not handle these assign statements well.
  - **Synthesis Limitations**: While synthesis should ideally eliminate these assign statements, it doesn't always succeed.
  - **v2lvs Conversion**: The v2lvs tool can convert some assign statements into *.connect commands.
  - **LVS Challenges**: Despite these conversions, LVS may still fail in certain scenarios, such as when an assign statement connects two inputs.
- **Solution:**
  - Remove assigns during init_design:
    - set init_remove_assigns 1
  - Just to make sure, remove assigns again after placement:
    - delete_assigns -add_buffer



Problem #3: Bus Notation

- **Issues:**





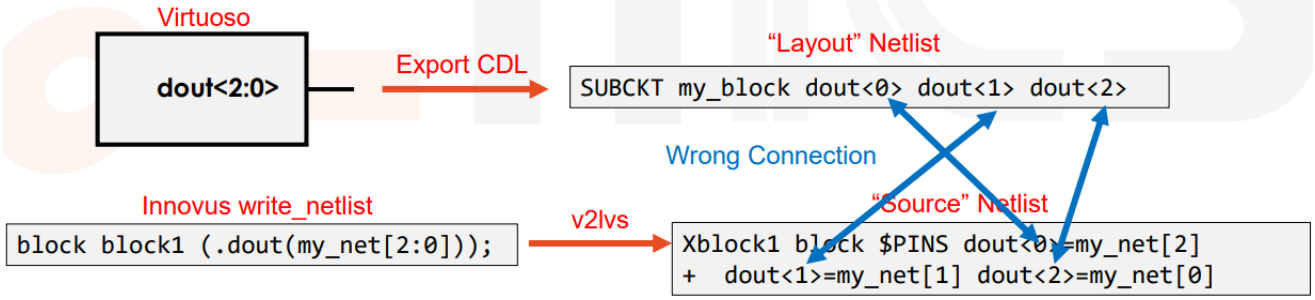
Solution:

- When exporting your CDL from Virtuoso, select "Map Bus Names from <> to [ ]"

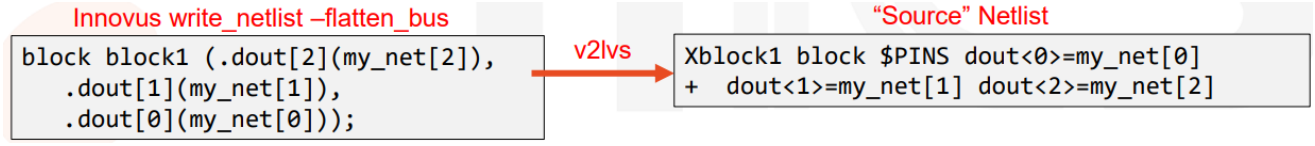
Problem #4: Flipped Busses

- Issue:
  - Digital tools use Verilog and often consider busses as multi-bit vectors.
  - Analog (circuit) tools use SPICE and don't necessarily use vectors.
  - Both languages support connectivity by position, which can lead to mismatches.

Example of Flipped Bus:



- Solution:
  - Connect independent signals and not busses
  - In Innovus, use the –flatten_bus option of write_netlist:
    write_netlist –phys –flatten_bus



Problem #5: Excluded Instances

- **Issue:**
  - Some "Physical Cells" do not have a corresponding CDL in the library.
  - These include (among others):
  - Fillers (not Well Taps!)
  - IO Fillers
  - Corner Ios
  - Bond Pads
  - Therefore, when running v2lvs, the tool cannot translate these to SPICE.

```
FILLER2 FILL123 (
    );
CORNER  TOP_LEFT_CORNER (
    );
```

- **Solution:**
  - Option 1: Use the –exclude_insts_of_cells option of write_netlist.
  - Option 2: Post-process them away (e.g., sed -i –e '/FILLER/,+1d')
  - Option 3: Create empty SUBCKT definitions

```
.SUBCKT FILLER2
.SUBCKT CORNER
```

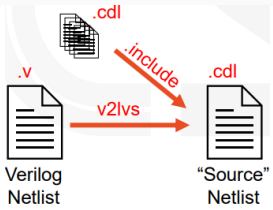
#Translating the Verilog netlist into SPICE:

Innovus has provided us with a Gatelevel Verilog netlist, but LVS runs on SPICE netlists. So, we need to create an include file that references the CDLs of:
The standard cells
The IOs
The Compiled Memories
Any other Hard Macros
Any hierarchical blocks that passed LVS standalone (but you should flatten these for final LVS)

```
.INCLUDE /path/to/Standard_cells.sp
.INCLUDE /path/to/IOs.sp
.INCLUDE /path/to/SRAM1.sp
.INCLUDE /path/to/Custom_block.cdl
```

The Mentor tool for converting Verilog to SPICE is called v2lvs:
v2lvs -sn -v my_verilog.v -o my_output_cdl.cdl -lsr my_includes_file.sp -s my_includes_file.sp

- **-s**: This switch is used to specify the source Verilog file that needs to be converted. It tells v2lvs which Verilog file to process.
- **-sn**: This switch is used to specify the name of the output netlist file. It allows you to define the name of the netlist file that v2lvs will generate.
- **-v**: This switch is used to specify the verbosity level of the output. It controls how much information v2lvs prints during its execution, which can be useful for debugging or understanding the conversion process.
- -lsr: Lists shorted resistors.

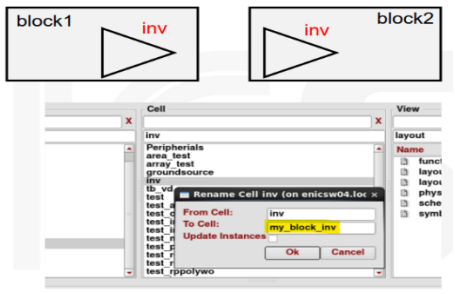


Issues:
Duplicate Subcircuits
Globals (e.g., Vendor provided IOs)
Bulk Connections (e.g., Standard Cells)
Ports that are shorted outside the block (e.g., on the board)

Problem #1a: Duplicate Custom Subcircuits

When creating a custom block, we may have a cell with the same name as a cell in some other block in the chip.

- **Solution #1:**
  In Virtuoso - give your custom instances a unique name by adding a prefix or suffix.



- **Solution #2:**

  Post-process your CDL to give subcircuits unique names by adding a suffix.

```
grep ".SUBCKT *" ${BLOCK_NAME}.sp | cut -d " " -f 2 > uniquify.lst
sed -i "/${BLOCK_NAME}/d" uniquify.lst
sed -i "s/.*/s\/&\/&_tile_${BLOCK_NAME}\/g/g" uniquify.lst
sed -f uniquify.lst ${BLOCK_NAME}.sp > ${BLOCK_NAME}.unique.sp
```

Problem #1b: Duplicate Digital Subcircuits

When running a bottom-up hierarchical flow, the RTL or EDA tools may (will!!!) create modules with the same name.

- **Solution #1:**
  - Tell Genus to give your modules a unique name:

```
set_attr gen_module_prefix "my_block"
```

  - Tell Genus to rename modules before netlist export

```
foreach module [get_db modules] {
        set name [get_db $module .base_name]
        rename_obj $module "my_block_$name" }
```

  - Tell Innovus to change module names before netlist export

```
update_names -module -suffix/prefix “my_block”
```

- **Solution #2:**

  Post-process your CDL to give subcircuits unique names by adding a suffix.

Problem #2: GLOBALs

A global signal in SPICE (.GLOBAL) propagates to the entire design and takes priority over local signals with the same name.

To clarify this, if you have VDD as a global signal in a block CDL any internal net called VDD will be connected to this signal.

```
.GLOBAL VDD
SUBCKT annoying_block
M1 VDD VDD VDD VDD NMOS
.ENDS
```

```
.SUBCKT BLOCK VDD
M1 VDD VDD VDD VDD nmos
.ENDS

.SUBCKT my_chip VDD1 VDD2
XBLOCK1 BLOCK $PINS VDD=VDD1
XBLOCK2 BLOCK $PINS VDD=VDD2
XANNOYING annoying_block
.ENDS
```

- **Solution:**
  - Don't use GLOBAL signals!

    Example: IOs for 65nm
    Unfortunately, the IOs we have for 65nm do use global signals
    `.GLOBAL VDD VSS VDDPST POC`
    This is a huge pain in the neck

  - Removing the Globals isn't enough:
    - CDL → No port connections to the relevant subcircuits.
    - Verilog Netlist →Since the IOs don't have any port connections for the globals, the Verilog netlist is exported without these connections.
    - LEF → If the LEF would have power connections, then they would be connected in Innovus and exported in the Verilog netlist.
    - Modify the LEF and CDL of the IOs

Problem #3: Bulk Connections

- **Issue:**

  A transistor has a bulk terminal.
  The standard cell LEF may not have a bulk terminal
  → there is no connection to the bulk for gates in Innovus
  → the exported Verilog netlist has no bulk connections

  This leads to two major problems:
  The standard cell SPICE (CDL) views have to have bulk connections
  → They are incompatible with the gate instantiation in the Verilog netlist.
  If we would globally define a bulk connection to VDD/GND, this wouldn't support power domains, body biasing, special power nets.

Solution to problem #1:

Use the –addpin option in v2lvs:

```
v2lvs -sn -v my_verilog.v -o my_output_cdl.cdl -s my_includes_file.sp
–addpin VPW –addpin VNW
```

This adds a connection with the same name as the pin, i.e.:

```
XCELL INVX1 $PINS A=in Z=out VDD=VDD VSS=VSS VPW=VPW VNW=VNW
```

This is okay for a single bulk bias, but not if several bulk biases are used.

Solution to problem #2:

- Post-process the CDL to connect the correct VPW/VNW.
- But it is much better and safer to MODIFY THE LEF!

Problem #4: Shorted Ports

Sometimes ports with different names are connected to each other – either at the board level or even for macro-level LVS.

For example, if separate VSS bulk connections are used, but they are connected through the substrate or to propagate the VNW/VPW signals.

- **Solution #1:**
  Use the *.CONNECT statement
  *.CONNECT VDD VNW [Note that this connects VNW to VDD so that VDD propagates through the circuit. If you switch the order, VNW will propagate through and your circuit will not pass LVS!]

- **Solution #2 (recommended):**
  Make a Wrapper that connects the two nets and run LVS on the wrapper

#Extracting the LVS-ready Layout Netlist

Now that our "Source" Netlist is ready, we need to prepare the "Layout" Netlist. We start by exporting the layout from Innovus in the GDSII format:

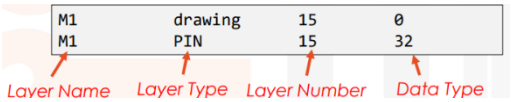
```
write_stream my_layout.gds -merge $ALL_GDS -mode NOFILL -map_file
$mapfile -unit 1000
```

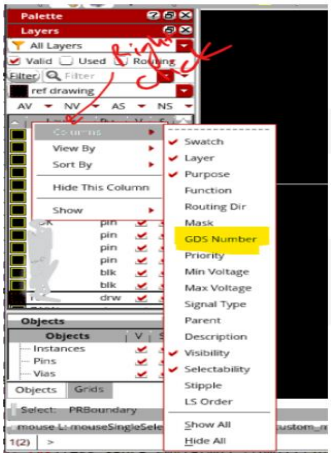
A few options to know about here…

• -merge $ALL_GDS: Merges the GDSII files of macros into the single output GDS file. Make sure you have all standard cells, IOs, etc.
• set_db write_stream_cell_name_prefix: Add a prefix for unique naming
• set_db write_stream_text_size: Sets the size of labels for readability

The Mapping File:

There are various types of GDS map files in EDA tools, which is confusing, but they all basically translate a layer name to its use and layer number.
The StreamOut MapFile used by Innovus and Virtuoso is a simple table:





It can be important to find the layer numbers for various purposes. One way is to turn on the GDS number in Virtuoso's LSW.

#Netlist Extraction:

So now we have the GDS and we have to extract the devices and connectivity to create the Layout Netlist. We can use Calibre to do the extraction:

```
calibre -hier -64 -hyper -turbo \
-spice my_layout_netlist.sp /path/to/runset.extract
```

The runset file tells the tool how to run the extraction.
First and foremost, this includes the path to the GDS file:

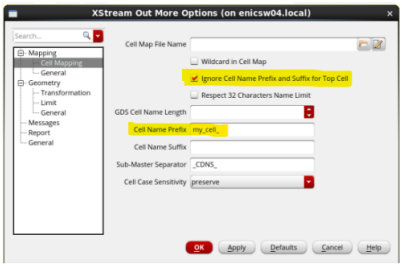
```
LAYOUT PATH "$MY_GDS"
LAYOUT PRIMARY "my_toplevel"
LAYOUT SYSTEM GDSII
```

Problem #1: Duplicate Instances

Merged GDS files have cells with the same name as other cells.

- **Solution for digital blocks:**
  Use set_db write_stream_cell_name_prefix to add a prefix to streamed cells.

- **Solution for custom blocks:**
  - Add a Cell Name Prefix on the XStream Out More Options form.
  - Make sure you also check the "Ignore Cell Name Prefix and Suffix for Top Cell" option



Problem #3: Missing Ports

Sometimes you are missing a label in your GDS. This will immediately cause a "port mismatch" in the LVS report

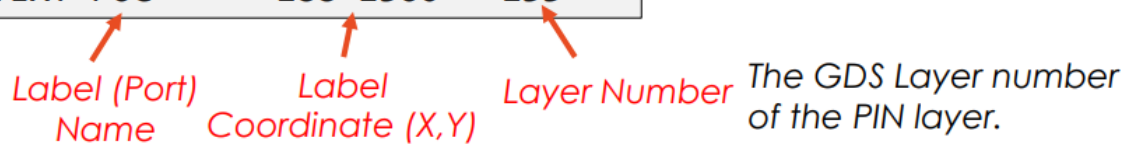
- **Solution #1:**

  Add it in Innovus. Not that easy, but the better solution.

- **Solution #2:**

  In the extraction runset file, use the LAYOUT TEXT command:

```
LAYOUT TEXT VDDPST  1037 1916   137
LAYOUT TEXT POC      188 1360   133
```



Problem #4: Multiple Labels

Multiple labels of the same name appear on the layout.

For example: VDD is on every VDD pad.

This is called a "Stamping Conflict" and will appear as a "Short Circuit" warning or error in the log file.

- **Solution:**
  - Use Virtual Connect Colon to connect labels with colons (e.g., VDD:)
  - Use Virtual Connect Name to connect labels with the same name (The option ? connects all nets with the same name)

```
VIRTUAL CONNECT COLON YES
VIRTUAL CONNECT NAME ?
```

Problem #5: Special Power Net Names

If you use a non-standard name (i.e., not VDD, VSS, GND…) to tap your bulks, Calibre will warn you that this the bulk is not connected to Power or Ground.

- **Solution:**

  Set the LVS POWER NAME and LVS GROUND NAME commands in the extraction runset file.

```
LVS POWER NAME "VDD" "VDD1" "VDDIO"
LVS GROUND NAME "VSS"
```

Problem #6: Blocks that aren't Ready

You want to start to setup and debug LVS, but you don't have all your IPs. Calibre will error out during extraction.

- **Solution:**

  Set the following command in the runset file:

  ```
  LAYOUT INPUT EXCEPTION SEVERITY MISSING_REFERENCE 1
  ```

Now, extraction will run, but you can't pass LVS, of course.
We will see how to exclude, filter, or box to try to get around this problem later.

Problem #7: Adding additional structures

You often need to add additional physical structures to the final GDS, such as a LOGO, Seal Ring, Dummy Fill, etc.

- **Solution #1:**

  Create a LEF for your physical structure and add it in Innovus.

- **Solution #2:**

  Merge the GDS of the physical structure with the toplevel GDS.

For example, using Calibre DRV:

```
calibredrv -a layout filemerge -append -createtop "my_toplevel" \
          -in my_logo.gds -out my_logo_for_merging.gds

calibredrv -a layout filemerge -append -topcell "my_toplevel" \ -in
          my_toplevel.gds -in my_logo_for_merging.gds \
                 -out my_toplevel_with_logo.gds
```

#Running LVS

So now we have the both the Source and the Layout netlists. We can use Calibre to run the comparison:

```
calibre -hier -64 -hyper -turbo \ /path/to/runset.compare
```

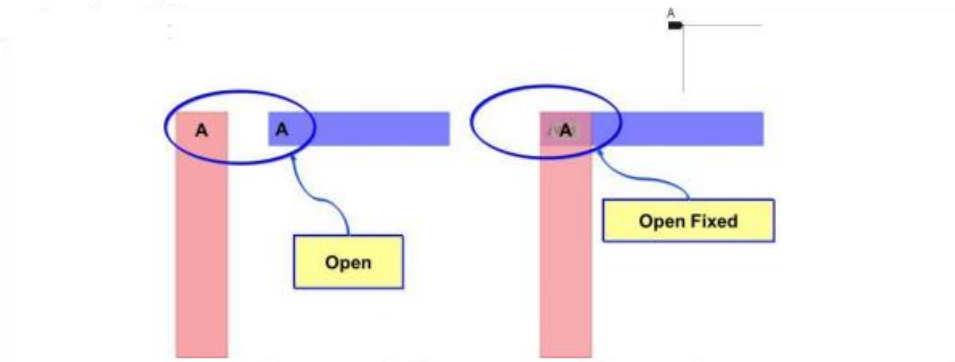
Here, too, we have a runset file tells the tool how to run the comparison.
First and foremost, this includes the paths to the two netlists:

```
LAYOUT PATH "$MY_LAYOUT_NETLIST"
LAYOUT PRIMARY "my_toplevel"
LAYOUT SYSTEM SPICE

SOURCE PATH "$MY_SOURCE_NETLIST"
SOURCE PRIMARY "my_toplevel"
SOURCE SYSTEM SPICE
```

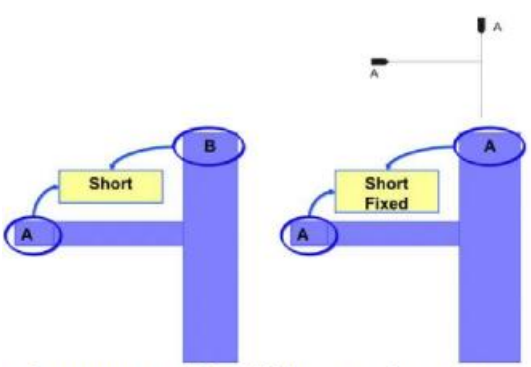
LVS checks examples:

Open Net Error:

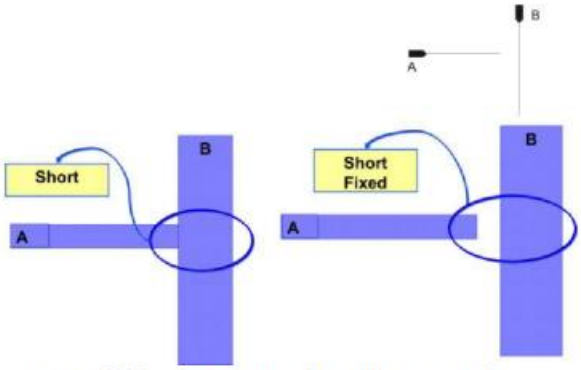


Same net is routed in two different metal layers but not connected

Short Net Error:



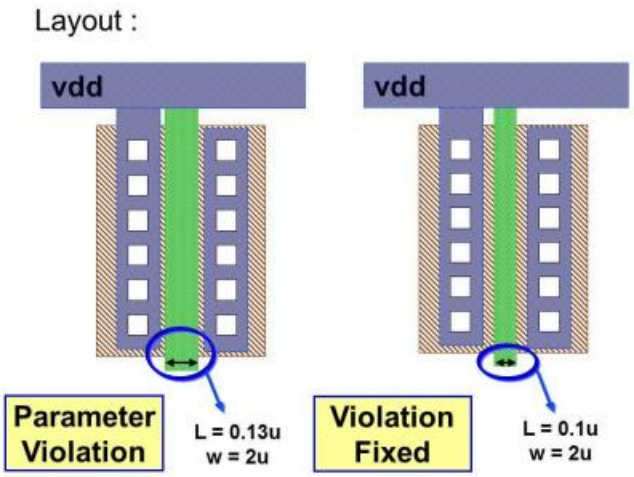
Same net with different pin names

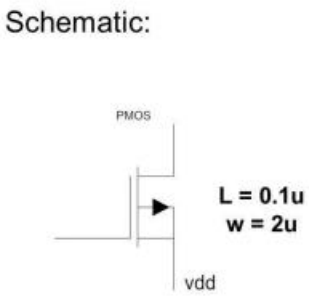


Two different nets shorting together

Extract Errors:

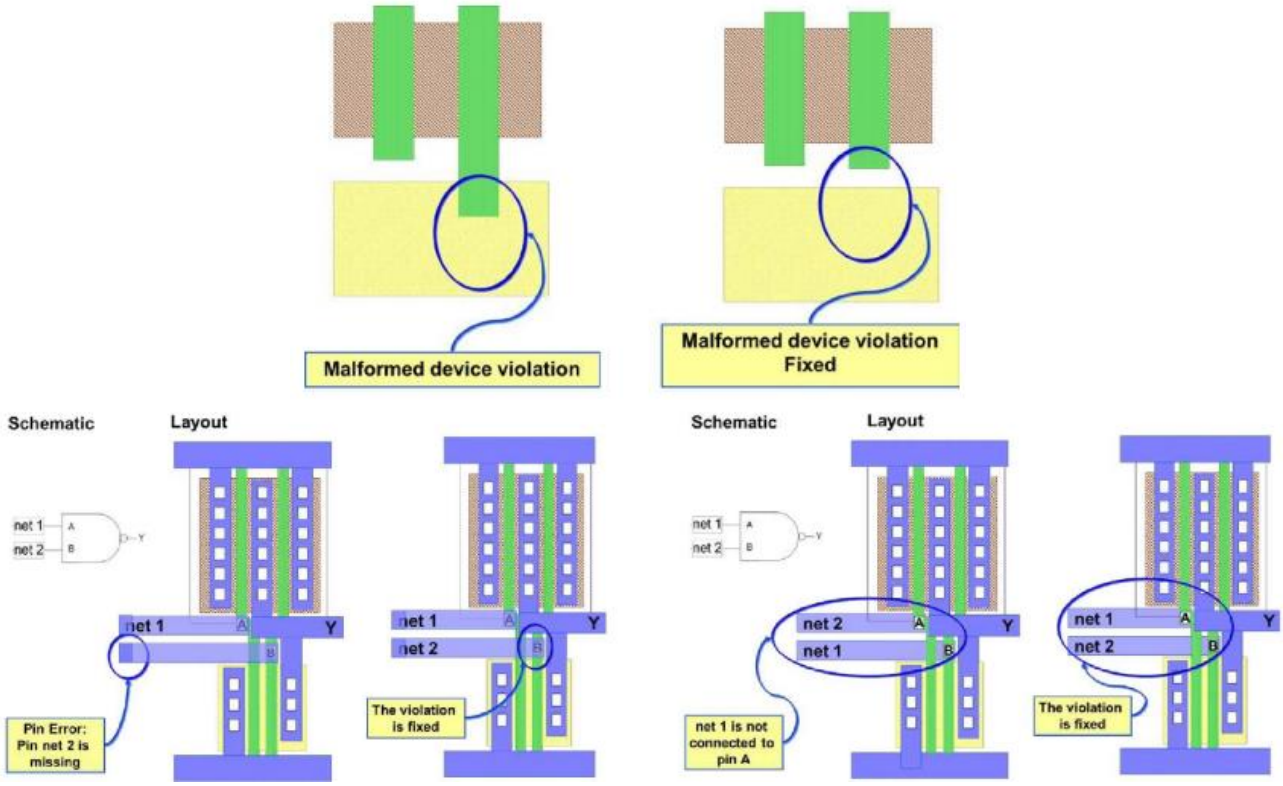
- Parameter Mismatch
- Device parameters on schematic and layout are compared
- Example: Let us consider a transistor here, LVS checks are necessary parameters like width, length, multiplication factor etc.





Compare Errors:

- Malformed Devices
- Pin Errors
- Device Mismatch
- Net Mismatch



#The LVS Results Database:

While running LVS in Calibre, the LVS result database usually dumped into a folder called "svdb". We can open it from Calibre REV:

```
calibre -rve svdb/
```

#LVS Issue Debugging:

- Port Comparison:
  - If your ports are not equivalent, don't look any further!
- Hierarchy Comparison:
  - Make sure lower level hierarchies pass LVS and ERC.
  - Make sure the connections to these hierarchies are correct.
- Bulk Connections:
  - The bulks are pretty clear – usually VSS for NMOS, VDD for PMOS.

- If some transistor in one of the netlists is connected to something else, that is probably a good place to start looking for problems…

- Device Connections:
  - "Best guesses" usually try to match devices by the most equivalent pin connections. If 3 out of 4 pins are connected correctly, try to look at the 4th pin.

- HCells:
  - A concept used in Layout Versus Schematic (LVS) verification to improve the efficiency and accuracy of the verification process. In essence, Hcells are repeated structures within a design that the LVS tool can recognize and handle more efficiently.
  - To get this to run, provide a file called "hcells" that details all hierarchical blocks (e.g. soft hierarchies, hard macros)
  - it's just is the name of the block… twice!

```
my_block1 my_block1
my_block2 my_block2
my_block3 my_block3
```

  - Then run LVS with the -hcells option:

```
calibre -hier -64 -hyper -turbo –hcells hcells.txt \
      /path/to/runset.extract
```

  - If you run LVS with the hcells option, each hierarchical block will be compared and reported separately. Obviously, all of the blocks must pass standalone LVS before integration!
  - So, if there is an error here, it probably is due to connections to the block and often VDD/GND connections, since signal connections shouldn't cause block level LVS to fail.

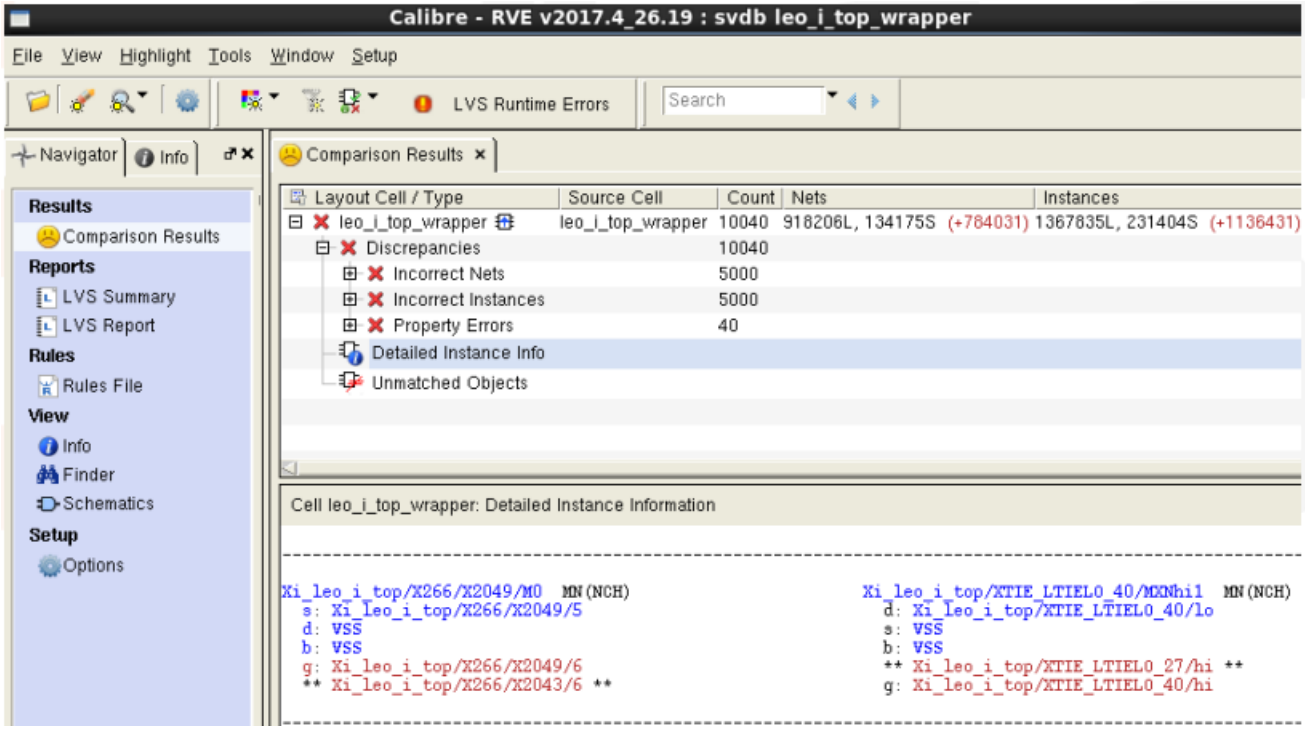
- Excludes, Filters, Blackboxes:
  - We can use the EXCLUDE, FILTER, or BOX commands to remove a block from our comparison
  - To understand the differences, read the Calibre manual, but the syntax is:

```
LVS SPICE EXCLUDE CELL LAYOUT MY_MISSING_BLOCK
LVS FILTER MY_PROBLEMATIC_BLOCK OPEN BOTH
LVS BOX MY_PROBLEMATIC_BLOCK
```

- Bulk Connections:
  - A good place to search for problems is in the bulk connections
  - Start by running DRC
    - DRC can highlight a lot of unexpected errors, such as NWELL overlaps, missing well taps, etc.
  - Then go to the ERC report
    - Are there warnings about short circuits or stamping conflicts with VDD/GND?
    - Check "soft connect" errors and highlight to see what's connected wrong.
  - Finally, look at the device connections in the LVS report
    - Your bulks should be connected to VDD (PMOS) or GND (NMOS)
    - If this is not the case (e.g., bulk is connected to net 1234), try to understand why.
  - Reason behind bulk connection issue:
    - Forgot to add fillers
    - Fillers added where they weren't supposed to be
    - Forgot to add Well Taps or wrong Well Tap cell used

- Wrongly connected Well Taps
- Macro overlaps

- Device Connections:
  - Look at the "detailed instance info" section of RVE.



ERC(Electrical Rule Check)

Ensure the reliability by checking for design rule violation related to power, ground, and signal integrity.

ERC checks are run to identify the following errors in layout:

- To locate devices connected directly between Power and Ground
- To locate floating Devices, Substrates and Wells
- To locate devices which are shorted
- To locate devices with missing connections

Input files:

- GDSII/OASIS layout file (e.g. design.gds)
- Technology rule deck (e.g. erc.rule)

```
// Example ERC Rule Deck

// Define voltage limits
VOLTAGE_LIMITS {
    VDD_MAX = 1.8V;
    VDD_MIN = 1.6V;
    VSS_MAX = 0.2V;
    VSS_MIN = -0.2V;
}

// Check for floating nodes
FLOATING_NODES {
    RULE "No floating nodes allowed";
    ACTION "Report";
}

// Check for short circuits
SHORT_CIRCUITS {
    RULE "No short circuits between VDD and VSS";
    ACTION "Report";
}

// Check for unconnected inputs
UNCONNECTED_INPUTS {
    RULE "All inputs must be connected";
    ACTION "Report";
}
```

- Power/Ground definition file **(Optional, depends on foundry)**
- Netlist file **(Optional, Used for comparison-based checks)**

- Use the following command to execute ERC:

```
calibre -erc -hier -turbo -64 -runset erc.runset
```

Explanation of Options:

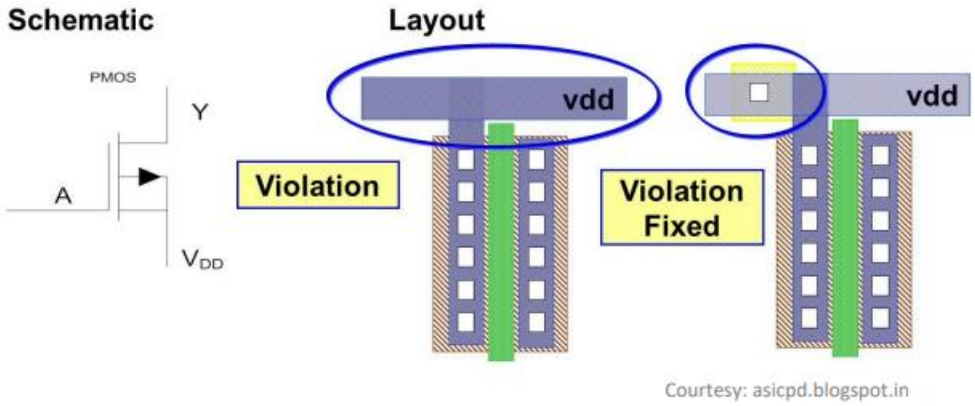
-erc → Run Electrical Rule Check
-hier → Run in hierarchical mode for efficiency
-turbo → Enable multi-threading for faster execution
-64 → Run in 64-bit mode
-runset erc.runset → Specifies the ERC runset file with settings

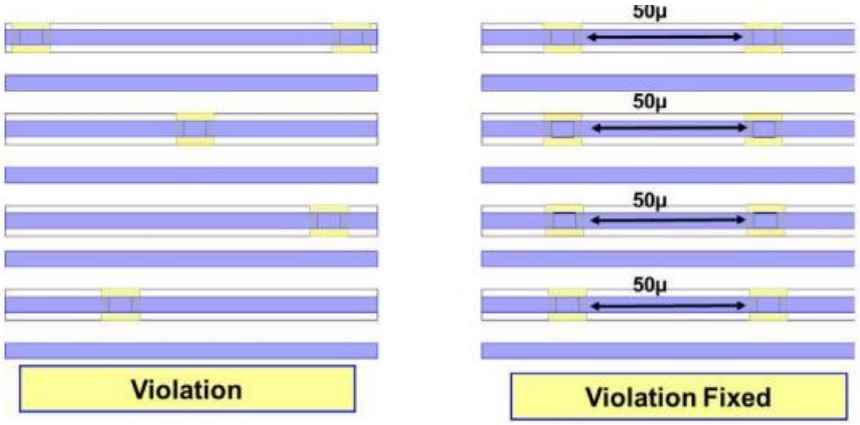
- Alternatively, run ERC with explicit input files:

```
calibre -erc -hier -turbo -64 -lvsdb design.lvsdb -gds
design.gds -rules erc.rule
```

-lvsdb design.lvsdb → LVS database from previous LVS run
-gds design.gds → Layout file in GDS format
-rules erc.rule → Rule file containing ERC checks

- Well Tap connection error: The Well Taps should bias the Wells as specified in the schematics. If there is no enough Taps for a given area then this error is flagged. Taps need to be placed regularly which biases the Well to prevent Latch-up.
    - e.g., In typical 90nm process the Well Tap Density Rule require Well-taps to be placed every 50 microns





#Some Frequent ERC error and fixes:

- Floating MOSFET Gates (Unconnected Gates)
    - Issue:
        - Some transistor gates are not connected to a valid signal. Causes leakage currents, unpredictable behavior, and high static power dissipation.
    - Fix:
        - Use a schematic vs. layout (LVS) check → Ensure that all transistor gates are connected.
        - Tie off unused inputs → Connect floating gates to VDD (pull-up) or VSS (pull-down) using tie-off cells.
        - Verify logic synthesis constraints → Ensure all required signals are routed properly.

- Power/Ground Shorts (VDD-VSS Shorts):
  - Issue:
    - VDD and VSS are shorted due to: Overlapping power/ground metal layers, incorrectly placed vias and Metal fill violations.
  - Fix:
    - Run Calibre RVE (or equivalent) to identify short location.
    - Inspect metal layers using a layout viewer → Ensure proper separation.
    - Use different routing layers for power and signal nets.
    - Check metal fill settings → Adjust to prevent shorts.

- High Resistance Power Paths (IR Drop Issues):
  - Issue:
    - Thin metal power routes lead to excessive IR drop, affecting circuit performance.
  - Fix:
    - Increase power strap width in critical areas.
    - Add more vias between metal layers to reduce resistance.
    - Use lower resistance metal layers (M4/M5 instead of M1/M2) for power distribution.
    - Run dynamic IR drop analysis to locate weak power delivery regions.

- Open Circuit (Disconnected Nets):
  - Issue:
    - Some signal nets or power rails are not connected, leading to functional failures.
  - Fix:
    - Check LVS results → Identify missing connections.
    - Manually route missing nets or use ECO fixes.
    - Run a connectivity check to ensure all standard cells are linked to power/ground.

- Electrostatic Discharge (ESD) Violations:
  - Issue:
    - Some input/output pins lack ESD protection diodes, risking chip failure.
  - Fix:
    - Check IO pads to ensure ESD diodes are present.
    - Follow foundry-specific ESD guidelines.
    - Use ERC checks for ESD protection in the rule deck.

- Latch-Up Risks (P-N Junction Shorts):
  - Issue:
    - Improper well tap connections can cause latch-up, leading to excessive current flow.
  - Fix:
    - Ensure well taps are properly placed to avoid parasitic thyristor effects.
    - Use guard rings around sensitive areas.

        - Increase well resistance by modifying doping parameters.

- Floating Metal (Unconnected Metal Segments)
    - Issue:
        - Isolated metal pieces cause antenna effects and charge accumulation.
    - Fix:
        - Ensure floating metals are connected to a known potential (VDD/VSS).
        - Run DRC/antenna rule checks to detect isolated metals.
        - Use dummy metal fills correctly to prevent charging issues.

Antenna Checks

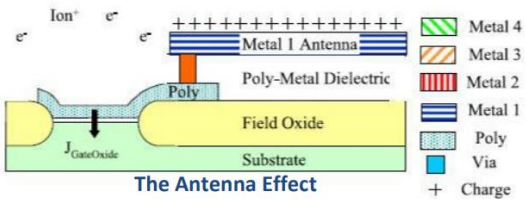
The antenna effect, also known as plasma-induced gate oxide damage, occurs during the plasma etching process used to create metal interconnects. During this process, charges can accumulate on the metal interconnects. If these interconnects are connected to the gate of a transistor, the accumulated charge can discharge through the thin gate oxide, potentially causing damage. This phenomenon occurs during process, so also known Process Antenna Effect (PAE).

Reason for Antenna Effect:

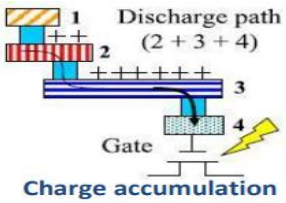
- Glow discharge during Plasma etching results in electric charging, which when occurred in conductive layer leads to Antenna effect thus termed Plasma Induced/ Process-Induced damage (PID)
- Charging occurs when conductor layers not covered by a shielding layer of oxide are directly exposed to Plasma
- During process like soldering the chip is protected with some shielding
- But during fabrication there is no such protection & will lead to Antenna effect
- For Aluminum based process PAE is prominent at Etching stage and for copper-based process PAE is prominent at Chemical-Mechanical Polishing (CMP) stage
- If the area of a higher metal layer connected to the Gate through lower metal layer/ layers, then the charge of higher metal layer got added to the lower metal layer which can also cause PAE called Accumulative Antenna Effect

Impact in the design:

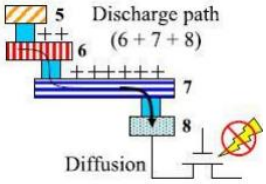
- If the area of the layer connected directly to the Gate the static charges are discharged through the Gate, the discharge can damage the oxide that insulates the gate and cause the chip to fail.
- Fowler-Nordheim (F-N) tunneling current will discharge through the thin oxide and cause damage to it.



The Antenna Effect



Charge accumulation & discharging on Poly

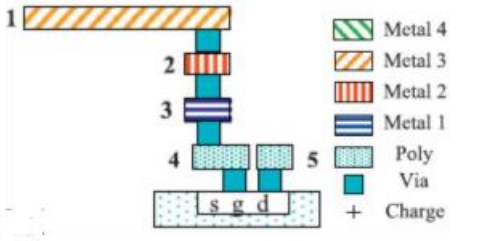


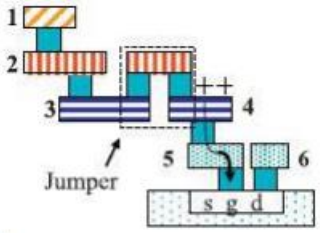
Charge accumulation & discharging on Diffusion

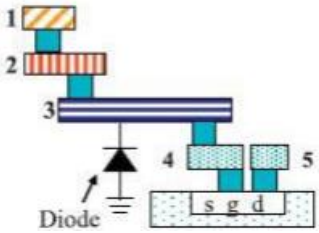
How to check antenna violations:

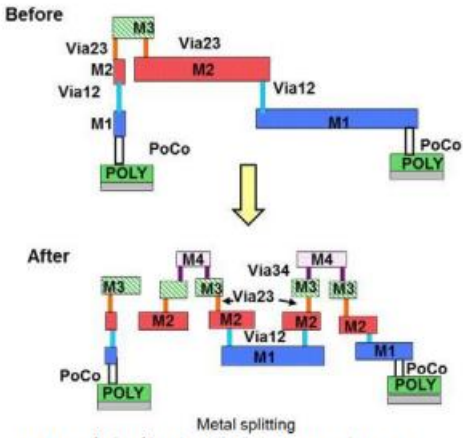
- During the place and route flow. But this is not sufficient due to missing information in the LEFs!
- Using the DRC tool with a special runset.

- **Solution to Antenna violations:**
  - Assigning higher metal layers for routing:
    - Higher metal layers will not be connected directly to the Gate Connect various metals through Via connections.
  - Inserting Jumpers:
    - If PAE is in lower layers then PAE can be reduced by connecting it to higher layers through Jumpers.
    - Jumpers will reduce the peripheral metal length, which is attached to the Gate
  - Connecting Antenna diode:
    - If it is in higher layers, Jumper won't be a solution, hence need diodes
    - As soon as extra charge is induced onto metal/ poly the diode diverts the extra charges to the substrate But for buffer insertion higher metal layers has to come to lower metal layer (M1 or M2) to connect to pins of buffer and go back and also there may not be enough place for buffer insertion
    - After routing only we go for antenna check, so Buffer insertion may lead to congestion and DRC violations

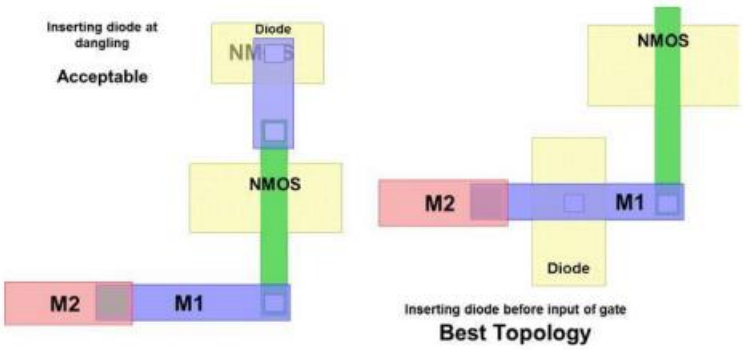




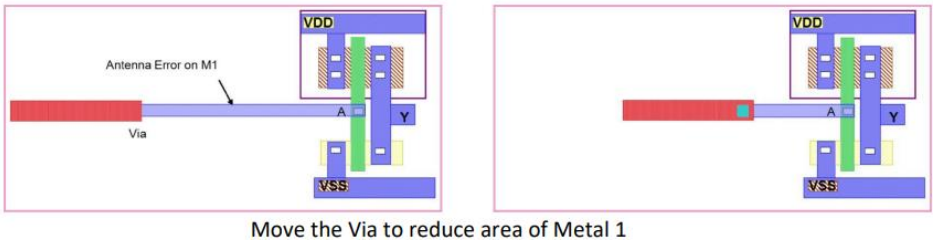




Metal Splitting: Connecting to higher Metal Layers



Diode placed at the unconnected end of gate (optional due to resistance of poly)



Move the Via to reduce area of Metal 1

Design Rule Check (DRC)

Design Rule Check (DRC) is the process of checking physical layout data against fabrication-specific rules specified by the foundry to ensure successful fabrication. DRC ensures layout compliance with foundry design rules (minimum width, spacing, enclosure, etc.). It is performed at the signoff stage to verify manufacturability before tapeout.

Input files:

- Final layout file → design.gds (generated after detailed routing).
- Technology file & rule deck → drc.rule (provided by the foundry).
- Netlist (optional) → Used for cross-verification with LVS.

Run DRC Check:

```
calibre -drc -hier -turbo -64 -rules drc.rule -gds design.gds
```

-drc → Enables DRC mode.
-hier → Runs a hierarchical check (for faster execution).
-turbo -64 → Runs in 64-bit turbo mode.
-rules drc.rule → Uses the DRC rule deck.
-gds design.gds → Specifies layout file.

We can filter out DRC errors that are irrelevant at this stage using the DRC UNSELECT CHECK command in rule file. E.g.: `DRC UNSELECT CHECK "M1.DN.1"`

Output Files:

drc.rpt → Detailed report of DRC violations.
drc.db → Database for GUI debugging.

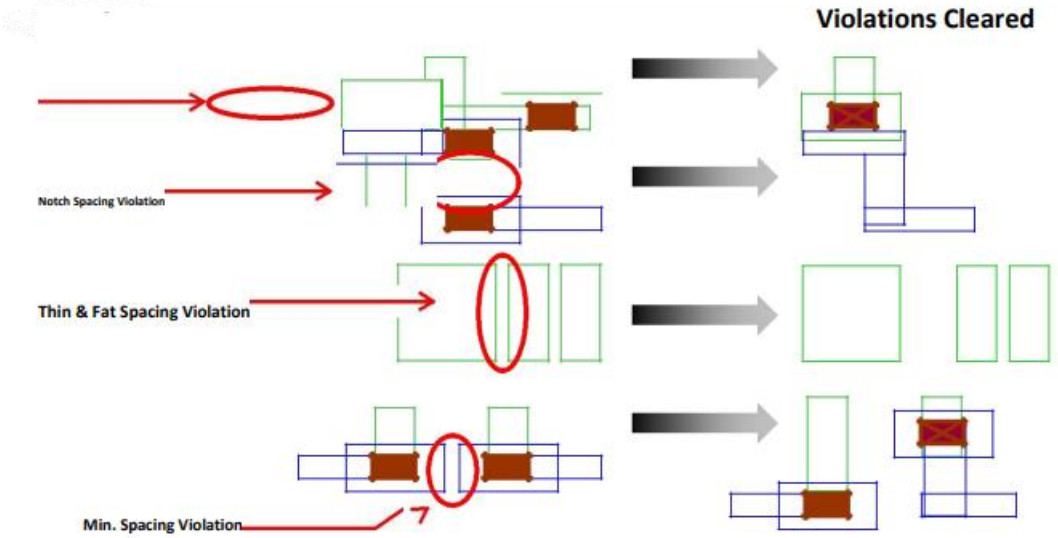
View DRC Violations in Calibre RVE:

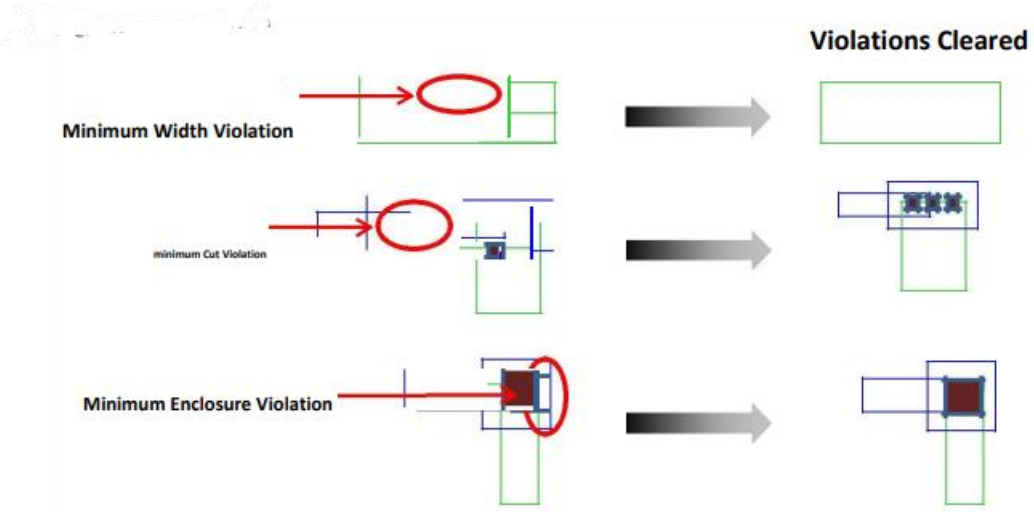
```
calibre -rve drc.db
```

Design Rule examples:

- **Maximum Rules:** Manufacturing of large continuous regions can lead to stress cracks. So 'wide metal' must be 'slotted' (holes)
- **Angles:** Usually only multiples of 45 degree are allowed
- **Grid:** All corner points must lie on a minimal grid, otherwise an "off grid error" is produced
- **Minimum Spacing:** The minimum spacing between objects on a single layer
- **Minimum Width**: The min width rule specifies the minimum width of individual shapes on a single layer
- **Minimum Enclosure/ Overlap:** Implies that the second layer is fully enclosed by the first one
- **Notch:** The rule specifies the minimum spacing rule for objects on the same net, including defining the minimum notch on a single-layer, merged object

- **Minimum Cut:** the minimum number of cuts a via must have when it is on a wide wire.





Reference:

1. Physical Design Flow V: Physical Verification - VLSI Pro
2. Digital-on-top Physical Verification LVS and DRC using Innovus and Calibre [https://www.eng.biu.ac.il/temanad/files/2020/09/Full-Chip-DRC-LVS-slides.pdf]
3. Physical-verification-slides - VLSI Guru