Layout Verification Reporter (LVR) Usecases

Ramtera Inc

December 21, 2025


LVR
Layout Verification
Reporter

Contents

Copyright

Ramtera Design Automation, Inc. 16192 Coastal Highway, Lewes, Delaware 19958, County of Sussex, USA.

1 Application Notes

1.1 Running LVR Using Command Script (Method 1)

This method describes how to run LVR using a **command script file**. It is the most straightforward and automatable method of running LVR, suitable for batch processing and regression environments.

1.1.1 Step 1: Prepare the Command Script

Create a command script file named `command_script.cmd` with the following contents: Each command corresponds to a critical step in the LVS flow: tool setup, data loading, verification, and visualization.

1.1.2 Step 2: Run LVR with the Script

Execute the command script by running the following in the terminal:

% LVR command_script.cmd

LVR will sequentially execute all the instructions defined in the script.

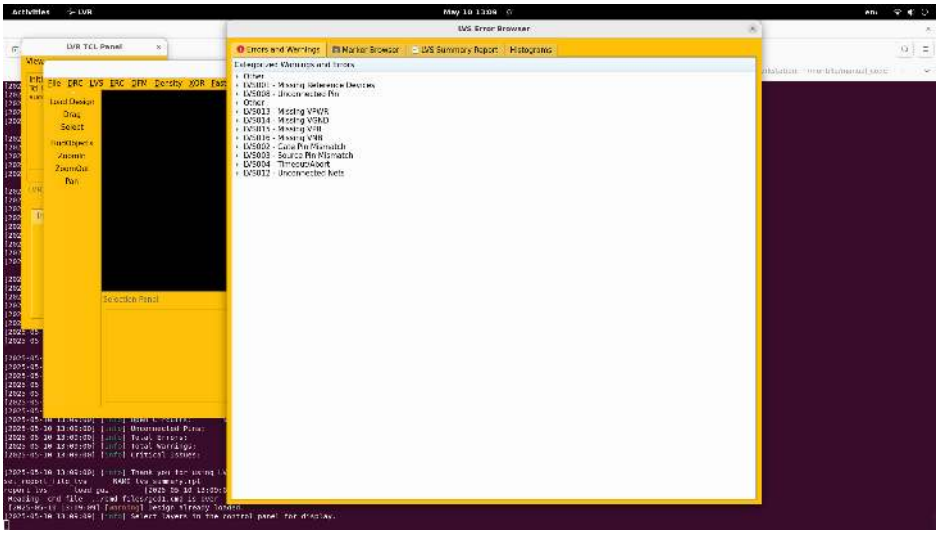
1.1.3 Step 3: Open GUI and Review Results

Once the GUI launches, use the tabs to review results:

- **Errors and Warnings** – Categorized issues such as missing nets, pin mismatches, etc.
- **Marker Browser** – For geometric location of errors
- **LVS Summary Report** – Overview report from `lvs_summary.rpt`
- **Histograms** – (if enabled) Error distributions

```
# initiatize tool DB
############################
init_tool_db LVS

# set techonlogy
############################
set_technode sky130

# set top module
############################
set_top gcd1_sky130hd

# set number of cores
############################
set_num_cores 10

# read the rule file and gds
############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db "LVS"

############################
# LVS settings
############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 1
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
############################
lvs gcd1_sky130hd gcd1_sky130hd.sch

############################
# set and generate report file
############################
set_report_file_lvs lvs_summary.rpt
report_lvs

############################
# load gui
############################
load_gui
```

Figure 1: Example command file to run LVR



Screenshot of the gui after the tool command load_gui is called.

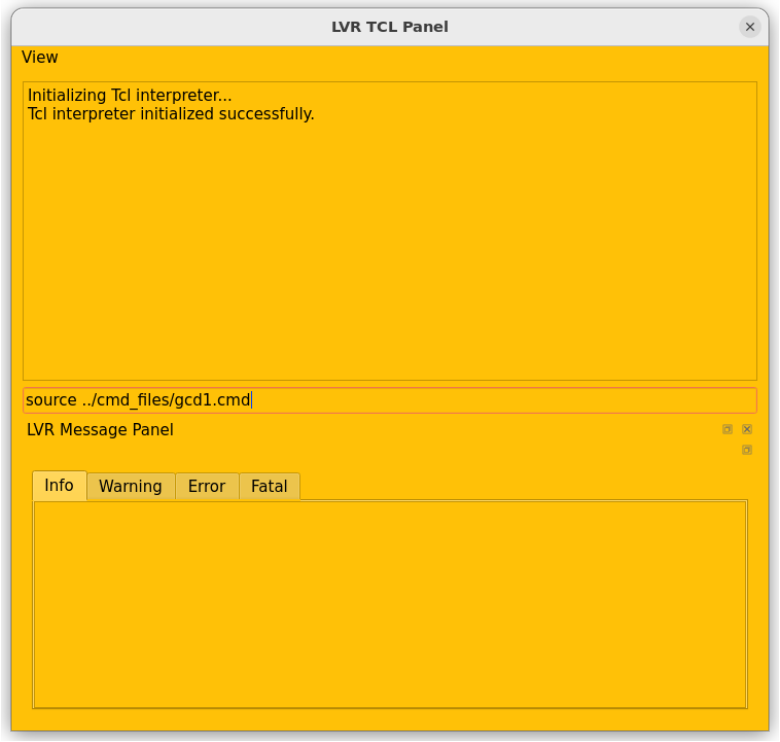
Sample categorized messages include:

- `LVS001` – Missing Reference Devices
- `LVS008` – Unconnected Pin
- `LVS013{LVS016` – Missing Power Rails (VPWR, VGND, VPB, VNB)
- `LVS020{LVS012` – Gate/Source/Pin Mismatches and Unconnected Nets

1.1.4 Conclusion

This method is ideal for:

- **Headless runs** (no GUI)
- **Automated test benches**
- **CI/CD pipelines**



1.2 Running LVR Using Command Script (Method 2)

This method allows users to interactively run LVR using its graphical interface. Instead of supplying the command file as a command-line argument, the user launches the GUI and uses the built-in TCL interpreter panel to load the script.

1.2.1 Step 1: Prepare the Command Script

As shown in **Method 1**, prepare a TCL script file (e.g., `command_script.cmd`) that contains the sequence of LVR commands for initialization, setup, LVS, and report generation.

1.2.2 Step 2: Launch LVR Without Arguments

Start LVR directly without specifying any script file: This opens the full graphical interface of the tool with access to TCL scripting and visualization features.

1.2.3 Step 3: Source the Script from the TCL Panel

In the **LVR TCL Panel**, type the following command:

% source ../cmd_files/gcd1.cmd

Then press `Enter`. The TCL interpreter will begin executing the script and display real-time status messages.

1.2.4 Step 4: Load Design in Layout Viewer

Once LVS run is complete, click on `Load Design` in the toolbar. This opens the layout view, allowing you to browse layers, navigate geometry, and examine graphical error markers.

1.2.5 Step 5: Open the Error Browser and Review Reports

Navigate to the **LVS Error Browser** window. It contains multiple tabs:

- **Errors and Warnings** – Lists categorized LVS issues.
- **LVS Summary Report** – Displays report headers, host, runtime, memory, and result.
- **Marker Browser** – Allows navigation to error markers.
- **Histograms** – Summarizes error distributions (if enabled).

1.2.6 Conclusion

This method is ideal for:

- Users who prefer a GUI-assisted workflow
- Interactive debugging of command files
- Quick script testing and visualization

With TCL scripting embedded directly into the GUI, this method provides a powerful mix of automation and interactivity.

1.3 Running LVR Using GUI Input (Method 3)

Open LVR by running the executable from your terminal:

% LVR

The tool will start and initialize its Tcl interpreter. The main window and TCL panel will appear.

1.3.1 Step 2: Open the Configuration GUI

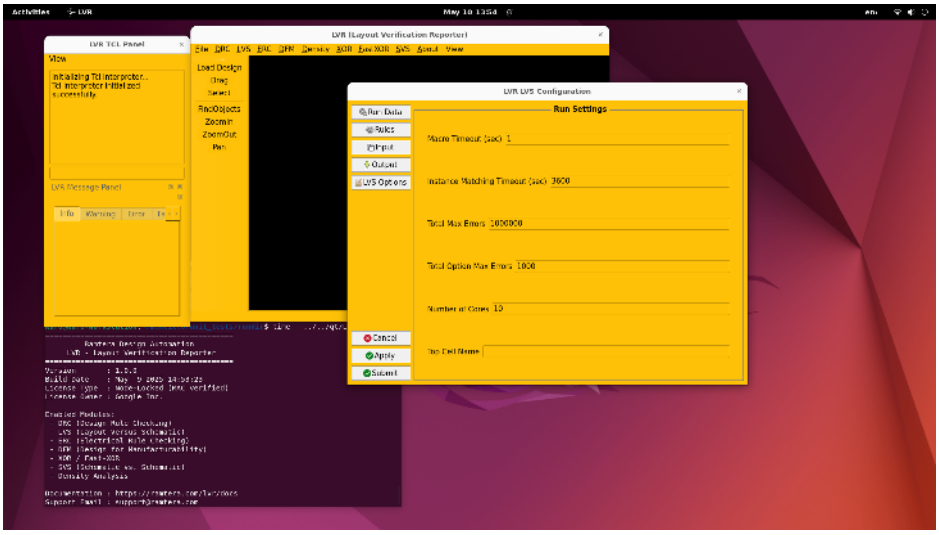
Click on the `LVS` menu and select the option to open the LVS configuration panel. This interface contains multiple tabs like:

- **Run Data** – Runtime parameters
- **Rules** – Paths to rule files
- **Input** – GDS, Verilog, CDL, SPICE
- **Output** – Output files
- **LVS Options** – Feature toggles and control flags

1.3.2 Step 3: Fill in All Required Fields

Manually enter all input values. Example fields include:

- `Macro Timeout`: 1 sec
- `Instance Matching Timeout`: 3600 sec
- `Total Max Errors`: 1000000
- `Total Option Max Errors`: 1000
- `Number of Cores`: 10
- `Top Cell Name`: (e.g., `gcd1_sky130hd`)



1.3.3 Step 4: Submit the Configuration

Click on the `Submit` button. LVR will begin executing the flow internally, based on the data provided. Progress and logs will be displayed in the message panel.

1.3.4 Step 5: Load Design and Debug

Once execution completes, click on `Load Design` to open the layout in the viewer. Then, open the **Error Browser** to inspect:

- Errors and Warnings
- LVS Summary Report
- Marker Browser
- Histograms (if applicable)

1.3.5 Conclusion

This GUI-based flow is ideal for:

- First-time users
- Manual setup or validation tasks
- Running smaller projects or one-off verifications

It provides the full functionality of LVR without requiring any scripting or command-line expertise.

2 Usecases

2.1 Test Case: gcd1_SVS001 (SVS)

Overview & Purpose

Context identification (0): From the name gcd1_SVS001, the tag “SVS” indicates this is a **Schematic vs. Schematic (SVS)** test.

Purpose (1.1): This test compares two netlist representations of the GCD design—SVS001A and SVS001B—to ensure structural/connection equivalence at the schematic level. It is intended to:

- verify that LVR correctly ingests two Verilog sources and performs side-by-side instance/net matching,
- surface connectivity differences (e.g., missing instances/nets, pin mismatches, opens),
- produce human-readable reports (Errors & Warnings, Marker Browser, Summary, Histogram) and enable fast debug in the GUI.

Figure 2 captures the high-level user flow for this use case (load rules → load SVS sources → run svs → inspect reports/GUI).

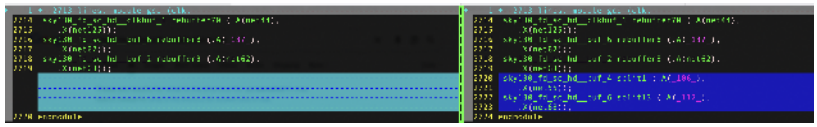


Figure 2: Use-case illustration for gcd1_SVS001 (SVS flow).

Command File (2, 2.1)

The file gcd1_SVS001.cmd (verbatim) used to run LVR for this SVS test:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db SVS

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
#read_gds ../gds/gcd1_sky130hd.gds
load_db SVS
no_compare 1

##############################
# LVS settings
##############################
lvs_set_datc_flow 1
```

```
#lvs_set_spice_file_name test
set_svs1_verilog ../verilog/sky130/gcd1_sky130hd_SVS001A.v
set_svs2_verilog ../verilog/sky130/gcd1_sky130hd_SVS001B.v
#set_svs1_verilog sv1
#set_svs2_verilog sv2
set_svs1_spice sp1
set_svs2_spice sp2
#set_svs1_spice ../spice/gcd1_sky130hd_SVS001A.sp
#set_svs2_spice ../spice/gcd1_sky130hd_SVS001B.sp
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
#lvs_check_open 1
#lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1

# call lvs
##############################
svs gcd1_sky130hd
#svs gcd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
```

```
############################
load_gui
```

Settings Explained (3)

Key settings from `gcd1_SVS001.cmd` and their roles:

- `init_tool_db SVS`: Starts LVR in **SVS** mode (schematic-vs-schematic comparison).
- `set_technode sky130`: Targets the Sky130 node; aligns rule/primitive expectations with the `sky130.lrf`.
- `set_top gcd`: Declares the top module for hierarchy resolution.
- `set_num_cores 1`: Single-core run (deterministic baseline).
- `set_die_area 0 0 106060 106060`, `set_num_regions 1`: Provides design bounds/regioning (useful for uniform defaults across flows).
- `read_lrf ../rule/sky130.lrf`: Loads LVR rule definitions (layer/primitive/lib expectations used by parsers and checks).
- `load_db SVS`: Finalizes DB prep for an SVS comparison run.
- `no_compare 1`: Keeps the flow strictly in the non-geometric/SVS path (bypasses layout-vs-schematic geometry compares).
- `lvs_set_datc_flow 1`: Enables the DatC-style netlist handling in the comparison pipeline.
- `set_svs1_verilog` / `set_svs2_verilog`: Supply the two Verilog sources to be matched.
- `set_svs1_spice` / `set_svs2_spice`: Placeholders for spice-side hooks; combined with `lvs_set_macros_cdl_file_name ../spice/sky130.cdl` to resolve library macros consistently.

- `svs gcd1_sky130hd`: Runs the comparison with project tag `gcd1_sky130hd`.
- `load_gui`: Opens the GUI (Error/Marker/Summary/Histogram tabs) for interactive debug.

How to Run (4)

Execute the test from a shell as:

```
% LVR gcd1_SVS001.cmd
```

This will produce the Error/Warning list, Marker browser, Summary (and Histogram if available) and launch the GUI for inspection.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 3 shows categorized messages. Typical items visible in this run include:

- *DB_WARN*: e.g., “number of cores is set as 1” (informational).
- *VER_WARN012*: nets like `VDD`/`VSS`/`net54`/`net67` reported open in one of the source files.
- *LRF_ERROR01*: several `.lrf` syntax errors with line numbers (points to rule-file clean-up needed).
- *SVS_ERR001/ERR005*: left/right matching not perfect; specific instance pins/nets (e.g., `clk`, `req_msg`, `resp_val`) remained unmatched.
- *SVS_ERR031*: mismatch in instance count (e.g., Left=359, Right=361).

Marker Browser (7–8). As indicated in Figure 4, the Marker Browser tab lets you load and navigate markers one-by-one (or clear them) for targeted debug of unmatched instances/nets.

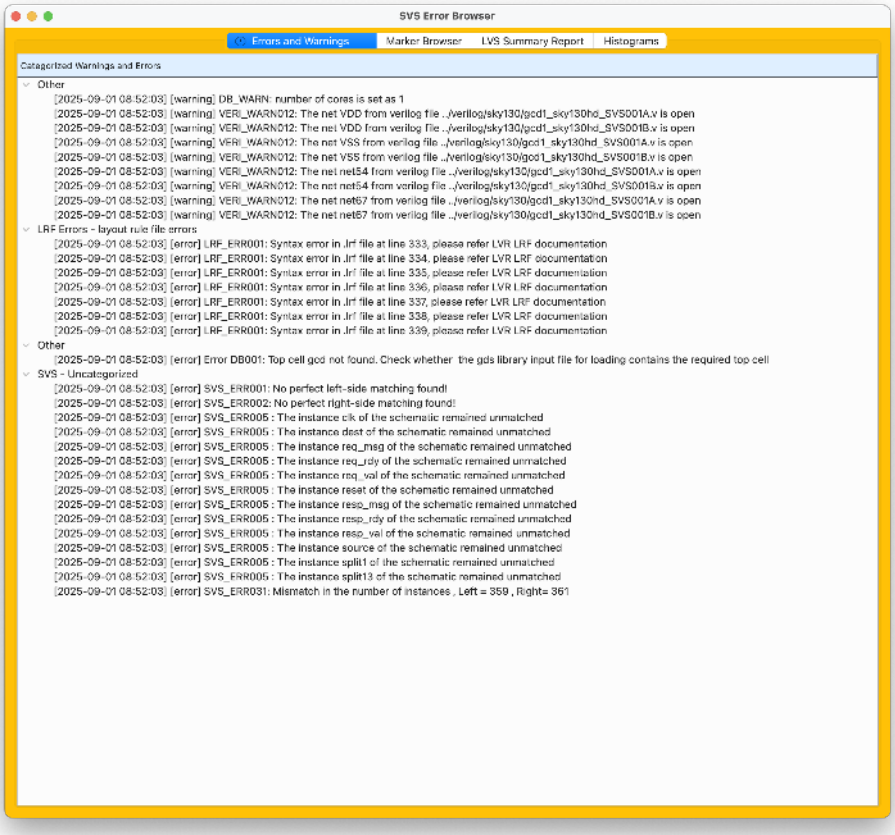


Figure 3: Errors and Warnings view for `gcd1_SVS001`.

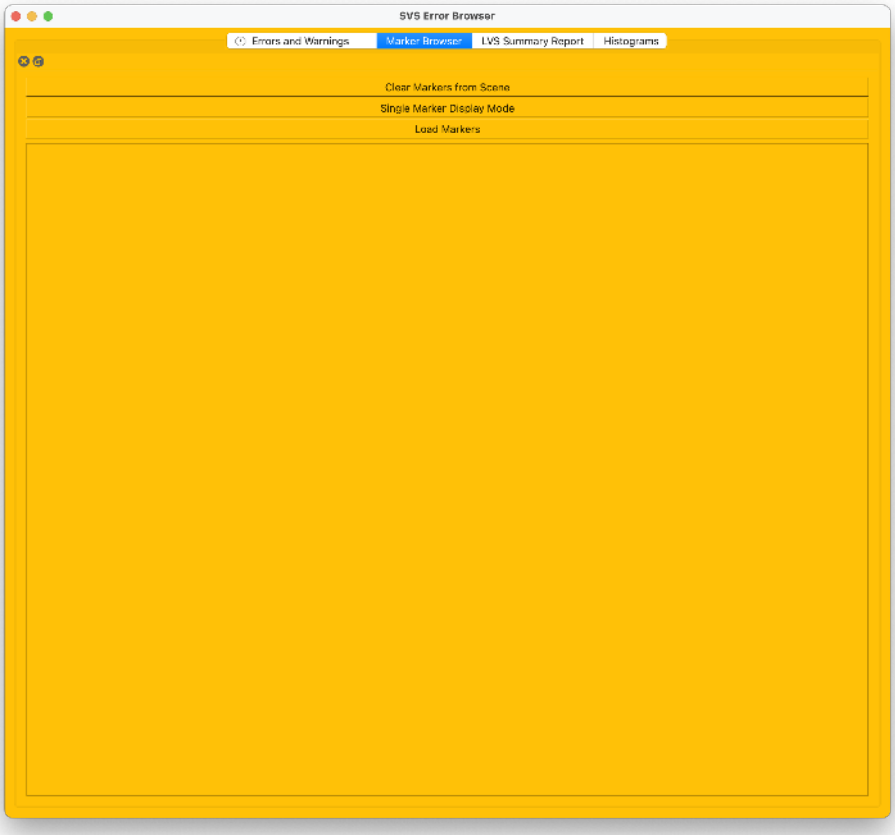


Figure 4: Marker Browser for `gcd1_SVS001` (load markers to inspect individual mismatches).

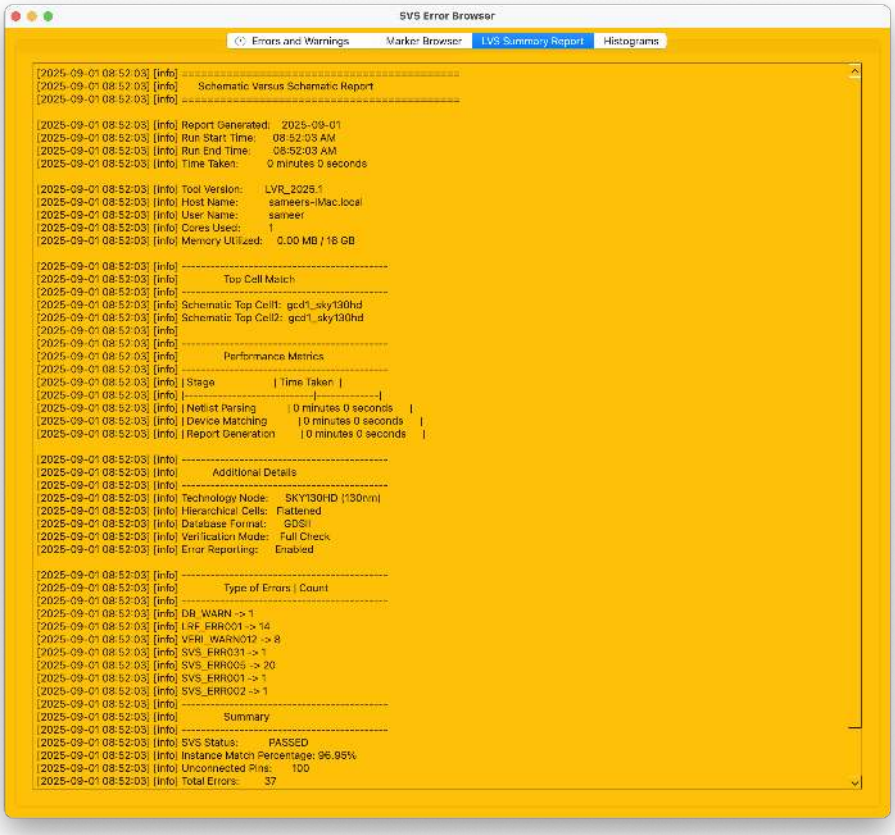


Figure 5: SVS Summary for `gcd1_SVS001`.

SVS Summary Report (9–10). Figure 5 summarizes the run:

- **SVS Status**: PASSED
- **Instance Match Percentage**: $\approx 96.95\%$
- **Total Errors**: 37
- Top-cell labels for both sources, timing breakdown for stages (Netlist parsing, Device matching, Report generation), node=SKY130HD (130nm), flattened DB, and "Full Check" verification mode.
- Counts by error type (e.g., DB_WARN, LRF_ERROR01, VER_WARN012, SVS_ERR001/2/5/31).

Notes for Debugging

The diff snapshot (not shown here) indicates additional instances present only on the right side (e.g., a split1 and split13 stage), which aligns with SVS_ERR031 and several SVS_ERR005 unmatched-instance reports. Start by reconciling those structural diffs, then re-run to confirm the instance-match percentage improves and total errors drop.

2.2 Test Case: gcd1_SVS002 (SVS)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag "SVS" in gcd1_SVS002 indicates a **Schematic vs. Schematic** comparison.

Purpose: Compare two schematic/netlist representations of the GCD design (SVS002A vs. SVS002B) at the Sky130 node to confirm structural connectivity equivalence. This test exercises LVR's SVS flow: rules loading, twin-netlist ingestion, instance/net matching, and GUI-assisted debug (errors, markers, summary, histogram).

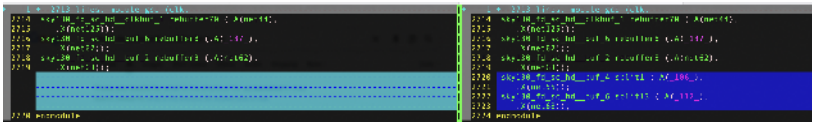


Figure 6: Use-case illustration for `gcd1_SVS002` (SVS flow).

Command File (2, 2.1)

Verbatim copy of `gcd1_SVS002.cmd`; these are the commands that run LVR for this test:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db SVS

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
```

```
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db SVS
no_compare 1

##############################
# SVS settings
##############################
lvs_set_datc_flow 1
#lvs_set_spice_file_name test
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_SVS002A.v
#lvs_set_verilog_netlist2_file_name ../verilog/sky130/gcd1_sky130hd_SVS002B.v
set_svs1_verilog ../verilog/sky130/gcd1_sky130hd_SVS002A.v
set_svs2_verilog ../verilog/sky130/gcd1_sky130hd_SVS002B.v
set_svs1_spice sp1
set_svs2_spice sp2
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
#lvs_check_open 1
##lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1

# call lvs
##############################
svs gcd1_sky130hd
svs gcd1_sky130hd

##############################
```

```
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

Settings Explained (3)

Key switches and why they matter:

- `init_tool_db SVS` initializes the database explicitly for **SVS** comparison mode.
- `set_technode sky130` selects Sky130; aligns parsing/macro expectations with `../rule/sky130.lrf`.
- `set_top gcd1_sky130hd` defines the schematic top for hierarchy resolution.
- `set_num_cores 1` creates a deterministic single-core baseline.
- `set_die_area ...`, `set_num_regions 1` standardize design bounds/regions for uniform flows.
- `read_lrf ...` loads rule constructs; `read_gds ...` is present though geometry is not compared (SVS only), which is fine for unified project context.
- `load_db SVS` finalizes DB prep for an SVS run; `no_compare 1` keeps the run schematic-only.

- `lvs_set_datc_flow 1` enables DatC-style netlist handling.
- `set_svs1_verilog`/`set_svs2_verilog` point LVR at the two Verilog sources; `set_svs*_spice` placeholders and `lvs_set_macros_cdl_file_name ...` ensure library macro resolution.
- `svs gcd1_sky130hd` triggers the comparison; it appears *twice* here, which simply re-invokes the SVS run with the same tag.
- `load_gui` launches the GUI (tabs for Errors & Warnings, Marker Browser, Summary, Histogram) for interactive debug.

How to Run (4)

Execute from a shell:

```
% LVR gcd1_SVS002.cmd
```

This runs the SVS comparison and launches the GUI. Reports are available in the GUI tabs and as log/summary artifacts.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 7 shows categorized diagnostics. Notable items in this run:

- *VER_WARN012*: several nets (e.g., `VDD`, `VSS`, `net54`, `net67`) reported *open* on one side (`SVS002A`/`SVS002B` annotations).
- *LRF_ERROR01*: syntax errors around lines $\sim$333–339 in the `.lrf`; clean these to avoid cascading issues.
- *SVS_ERR001*: no perfect left/right match; *SVS_ERR005*: specific signals like `clk`, `req_msg`, `resp_msg`, `reset`, `req_val`, `resp_val`, `source`, `split13` remained unmatched.
- *SVS_ERR031*: instance-count mismatch (Left $= 360$, Right $= 361$).

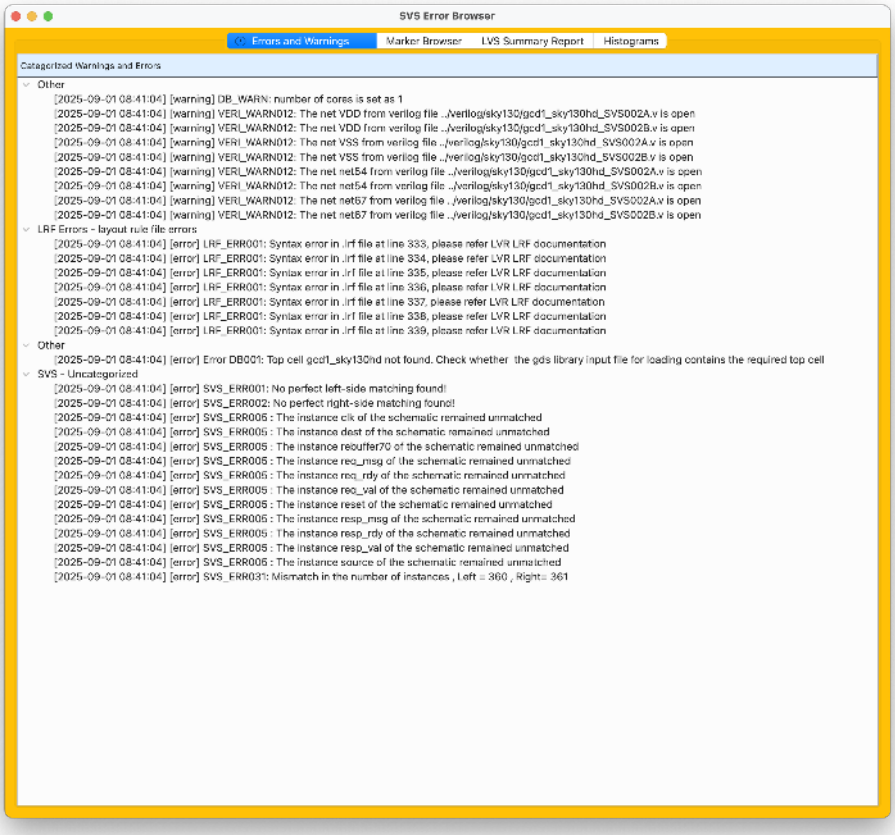


Figure 7: Errors and Warnings for `gcd1_SVS002`.

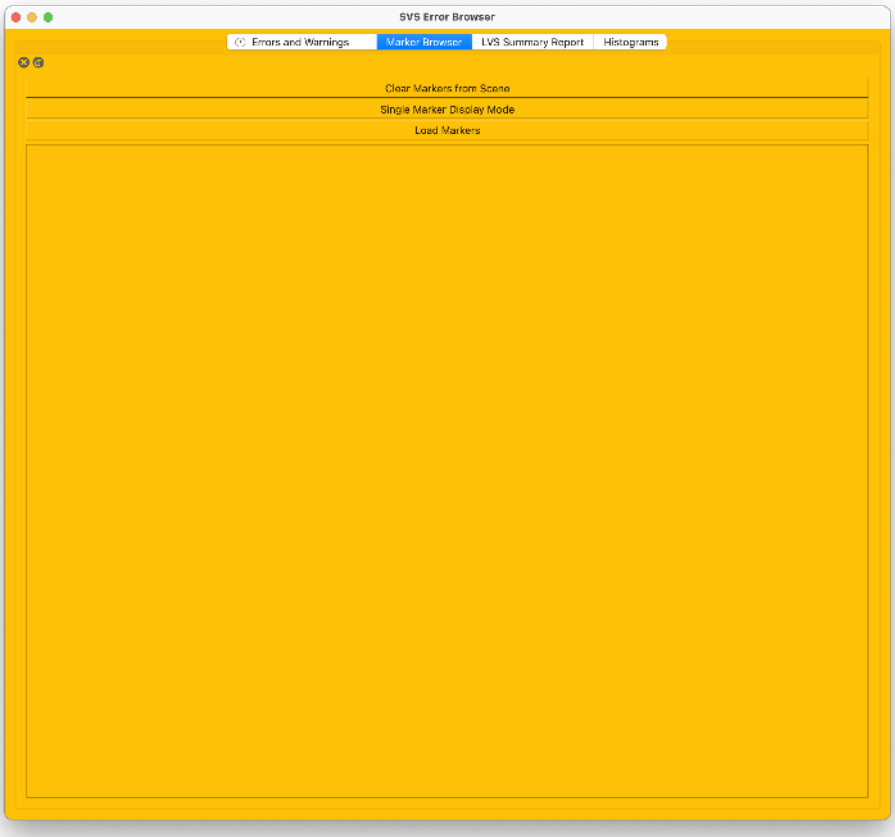


Figure 8: Marker Browser for `gcd1_SVS002` (load markers, step through mismatches).

Marker Browser (7–8). As shown in Figure 8, load markers to navigate instance/net mismatches one by one. Use single-marker mode for focused inspection.

SVS Summary Report (9–10). Figure 9 summarizes the run (example fields visible on screen):

- **SVS Status**: `PASSED`
- **Instance Match Percentage**: $\approx 97.23\%$

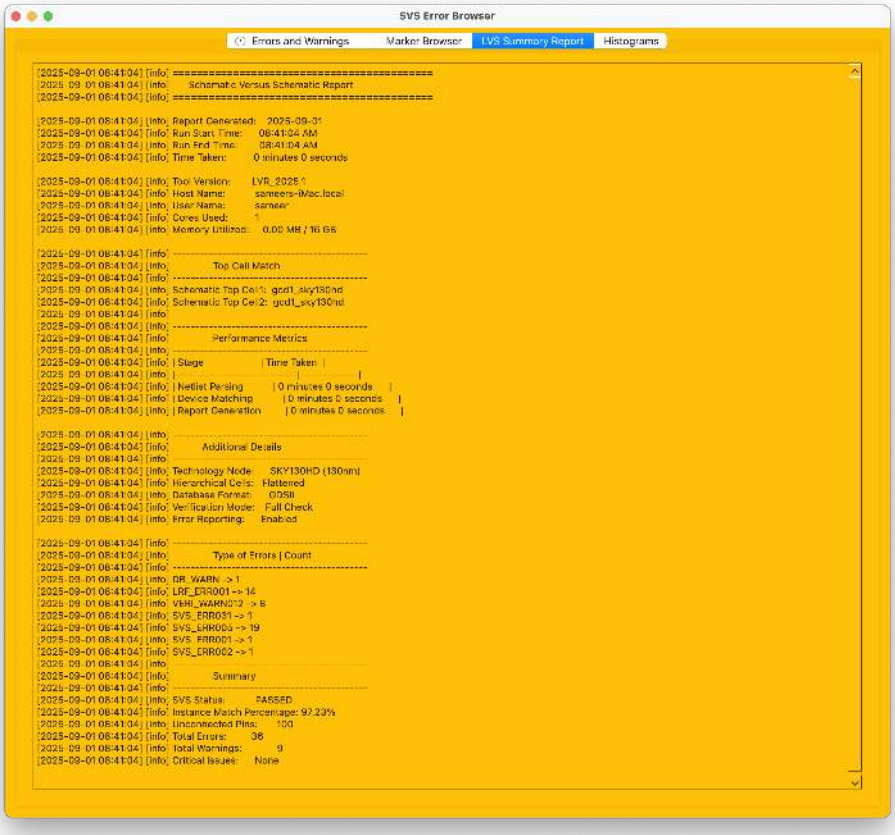


Figure 9: SVS Summary for `gcd1_SVS002`.

- **Total Errors**: 36; **Total Warnings**: 9; **Critical Issues**: none
- Node: SKY130HD (130nm), flattened DB, GDSII format, Full Check verification mode, error reporting enabled.
- Stage timings for Netlist Parsing, Device Matching, and Report Generation.

Quick Debug Note

The diff panel (not included here) points to extra right-side instances (e.g., a split stage), consistent with SVS_ERR031. Reconcile the structural delta and re-run; open-net warnings should drop if undeclared/unused pins are tied or removed consistently across both sources.

2.3 Test Case: gcd1_SVS005 (SVS)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag "SVS" in gcd1_SVS005 indicates a **Schematic vs. Schematic** comparison.

Purpose: Compare two Sky130 GCD schematic/netlist variants (SVS005A vs. SVS005B) to validate logical connectivity equivalence. This exercise stresses LVR's SVS flow: rule loading, twin-netlist ingestion, left/right instance & net matching, and GUI-assisted debug (Errors & Warnings, Marker Browser, Summary, Histogram). Figure 10 sketches the use-case pipeline: rules $\rightarrow$ SVS inputs $\rightarrow$ svs run $\rightarrow$ report inspection.

Command File (2, 2.1)

Verbatim listing of gcd1_SVS005.cmd; these commands run LVR for this SVS test:

```
# initiatize tool DB
```

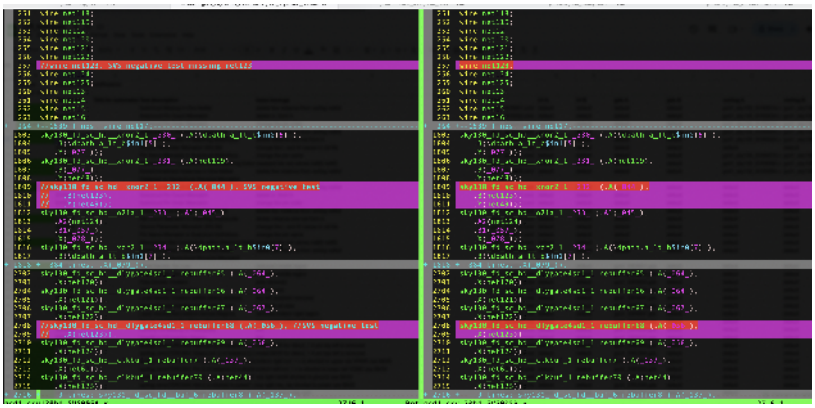


Figure 10: Use-case illustration for `gcd1_SVS005` (SVS flow).

```
##############################
#set_debug_mode_3049 0
init_tool_db SVS

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
```

```
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db SVS
no_compare 1

##############################
# SVS settings
##############################
lvs_set_datc_flow 1
#lvs_set_spice_file_name test
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_SVS005A.v
#lvs_set_verilog_netlist2_file_name ../verilog/sky130/gcd1_sky130hd_SVS005B.v
set_svs1_verilog ../verilog/sky130/gcd1_sky130hd_SVS005A.v
set_svs2_verilog ../verilog/sky130/gcd1_sky130hd_SVS005B.v
set_svs1_spice sp1
set_svs2_spice sp2
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
#lvs_check_open 1
#lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1

# call lvs
##############################
svs gcd1_sky130hd

##############################
```

```
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_svs

############################
# load gui
############################
#set_log_flush_time 1
load_gui
```

Settings Explained (3)

Key switches and intent:

- `init_tool_db SVS`: start LVR in **SVS** mode (schematic-vs-schematic).
- `set_technode sky130` and `read_lrf ../rule/sky130.lrf`: align parsing/macro expectations with Sky130 rules.
- `set_top gcd1_sky130hd`: declare schematic top for hierarchy resolution.
- `set_num_cores 1`: single-core, deterministic baseline; `set_die_area`/`set_num_regions`: standardize bounds/regioning.
- `read_gds ...`: loads layout context (useful for a unified project); `no_compare 1` keeps the run schematic-only.
- `lvs_set_datc_flow 1`: enable DatC-style netlist handling in comparison.
- `set_svs1_verilog` / `set_svs2_verilog`: point to the two Verilog sources to be matched.

- `set_svs*_spice` placeholders + `lvs_set_macros_cdl_file_name ...`: ensure macro/library resolution consistency.
- `svs gcd1_sky130hd`: execute the comparison; `load_gui` opens the GUI (tabs for Errors & Warnings, Marker Browser, Summary, Histogram).

How to Run (4)

Execute from a shell:

```
% LVR gcd1_SVS005.cmd
```

This launches the SVS comparison and opens the GUI. Artifacts appear in the GUI tabs and logs.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 11 shows categorized diagnostics. Highlights include:

- *VER_WARN012*: opens on nets like `VDD`, `VSS`, `net54`, `net67` (A/B suffix indicates which side).
- *LRF_ERROR01*: `.lrf` syntax issues around lines 333–339.
- *SVS_ERR001*: no perfect left/right match; *SVS_ERR005*: specific signals (e.g., `clk`, `dest`, `req_msg`, `resp_msg`, `reset`, `req_val`, `resp_val`, `source`) remained unmatched; also an unmatched instance `__232__`.
- *SVS_ERR031*: instance-count mismatch (Left $= 359$, Right $= 361$).

Marker Browser (7–8). As shown in Figure 12, load markers to navigate individual mismatches; single-marker mode helps focused inspection.

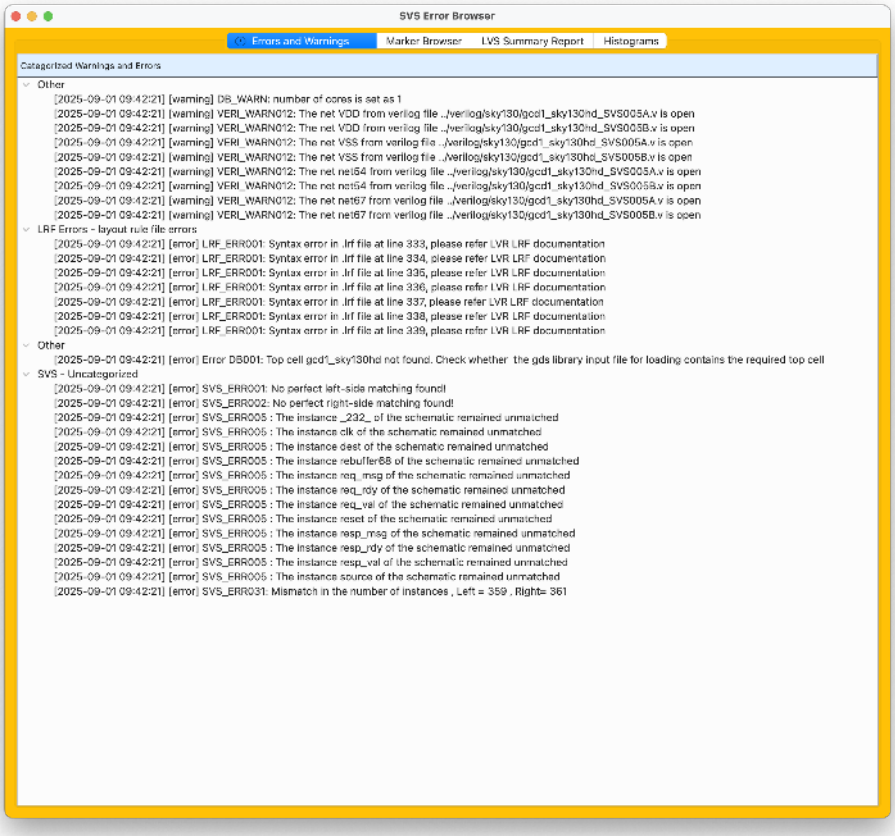


Figure 11: Errors and Warnings for `gcd1_SVS005`.

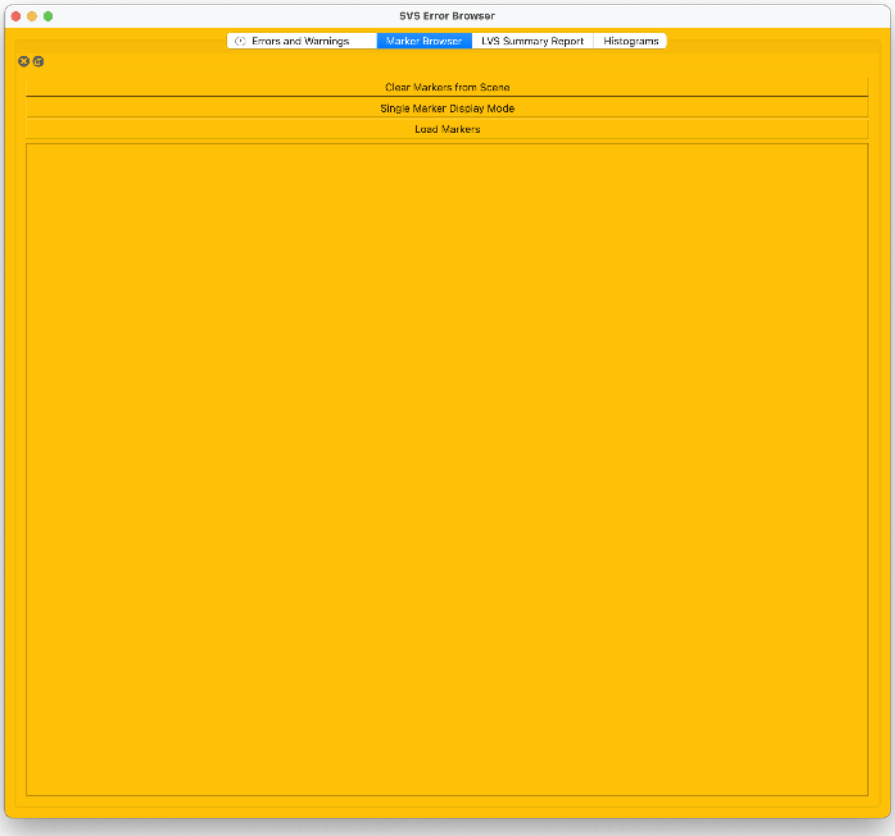


Figure 12: Marker Browser for `gcd1_SVS005` (load markers and step through mismatches).

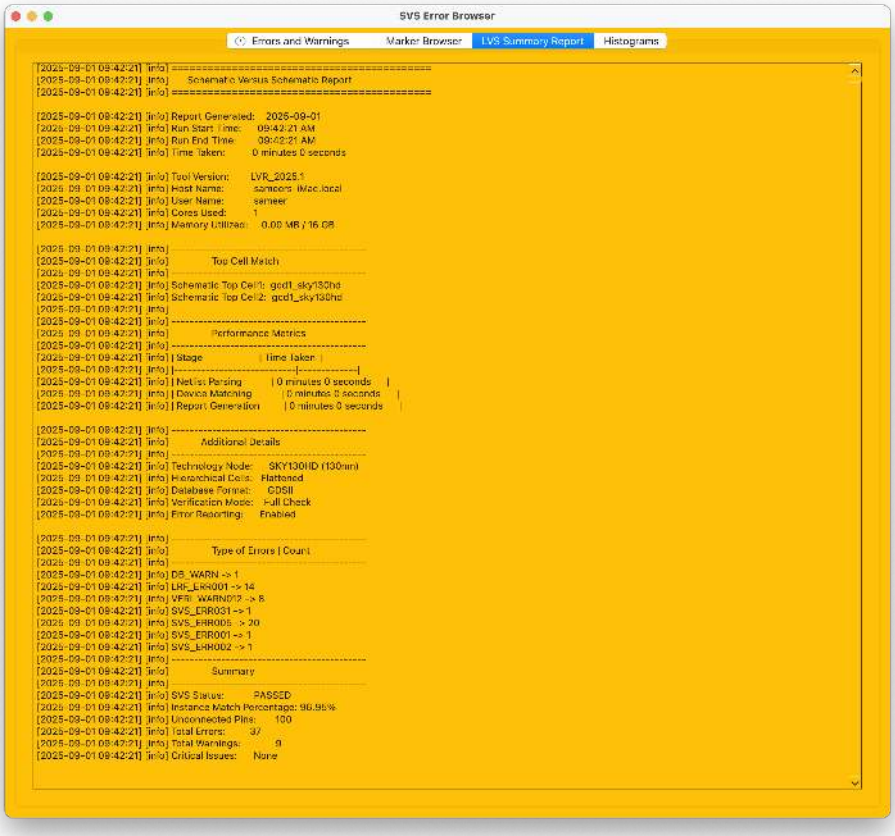


Figure 13: SVS Summary for `gcd1_SVS005`.

SVS Summary Report (9–10). From Figure 13: **SVS Status**=PASSED, **Instance Match** $\approx 96.95\%$, **Total Errors**=37, **Total Warnings**=9, **Unconnected Pins**=100; node=SKY130HD (130nm), flattened DB, GDSII, Full Check mode with timings reported for Netlist Parsing, Device Matching, and Report Generation.

Debug Note (diff context)

The provided diff snapshot for this test shows intentional negative edits (e.g., missing net123, extra buffers like rebuffer68, and an xnor2_1_232 instance on one side). These explain SVS_ERR031 (instance-count mismatch) and multiple SVS_ERR005 unmatched-signal reports. Reconciling those deltas (restore/tie the missing net, mirror added logic, or consistently remove across both sources) should raise the match percentage and reduce error counts.

2.4 Test Case: gcd1_ERC001 (ERC)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag "ERC" in gcd1_ERC001 indicates an **Electrical Rule Check**.

Purpose: Demonstrate ERC short detection on the Sky130 GCD layout when a signal is unintentionally connected to power. In this negative test, the *bottom-right* buffer instance (buf_1) has its output pin X shorted to the **upper VPWR rail** (Metal1). The note "use 68/20" refers to the author's layout guidance for reproducing/locating the condition in the editing grid. The goal is to verify that LVR:

- loads the Sky130 rule context and the test GDS,
- correlates layout nets and library macros against the schematic,
- detects power/ground shorts (e.g., VDD $\leftrightarrow$ signal nets on MET1),

- reports issues via Errors & Warnings, Marker Browser, Summary, and Histograms,
- and enables rapid root-cause debug in the GUI.

Figure 14: Use-case illustration for `gcd1_ERC001`: rules → GDS load → ERC run → report/GUI debug.

Command File (2, 2.1)

Verbatim listing of `gcd1_ERC001.cmd`; these commands run LVR for this ERC test:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
##############################
set_technode sky130
```

```
# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_ERC001.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
```

```
lvs_check_unconnected_pins 1

# call lvs
##############################
erc gcd1_sky130hd
#erc gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
load_gui
```

Settings Explained (3)

Key switches and intent:

- `init_tool_db ERC`: initializes the database in **ERC** mode (electrical connectivity sanity).
- `set_technode sky130` & `read_lrf ../rule/sky130.lrf`: select Sky130 rules and primitive/lib context.
- `set_top gcd1_sky130hd`: names the top cell for hierarchical resolution across layout/schematic.
- `set_num_cores 1`: deterministic single-core run baseline. `set_die_area`/`set_num_regions`: standard bounds/region setup.

- `read_gds ../gds/gcd1_ERC001.gds`: loads the edited GDS that contains the deliberate VPWR short at the buffer output.
- `load_db ERC` & `no_compare 1`: finalize DB and keep the flow in electrical checks (not geometric DRC).
- `lvs_*` toggles: even in ERC flow, LVR reuses LVS infrastructure for netlist/macros: DatC handling, macro CDL, port enablement, and optional open/short checks.
- `lvs_check_shorts 1`/`lvs_check_open 1`: enable short/open detection.
- `erc gcd1_sky130hd`: executes the ERC analysis.
- `load_gui`: launches the GUI with the tabs *Errors & Warnings*, *Marker Browser*, *ERC Summary Report*, and *Histograms*.

How to Run (4)

Execute from a shell:

```
% LVR gcd1_ERC001.cmd
```

This runs ERC and opens the GUI. Reports are available within the GUI tabs and associated logs.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 15 lists categorized diagnostics. Highlights visible include:

- database/tool warnings and layer-map not-found messages (ignored for run),
- `VER_WARN012`: `VDD`/`VSS`/`net54`/`net67` open warnings (from verilog context),

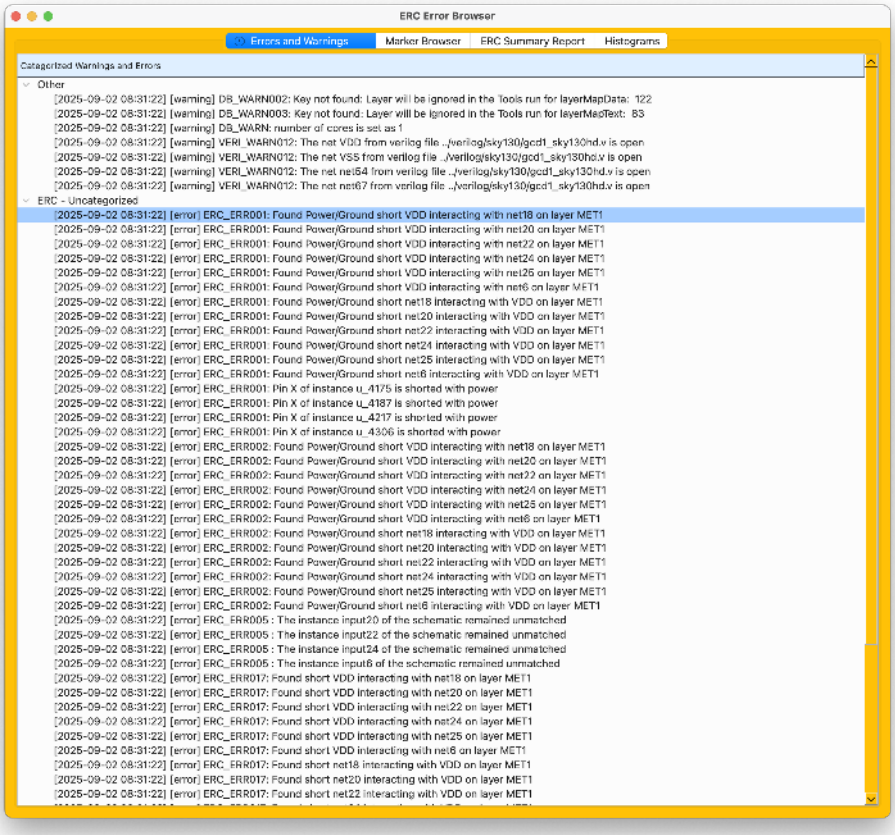


Figure 15: Errors & Warnings for `gcd1_ERC001`.

- numerous ERC_ERR001/ERC_ERR002: **Power/Ground shorts**—e.g., VDD interacting with signal nets net18, net20, net22, net24, net25, net6 on **MET1**.

These align with the intended defect (buffer X node shorted to VPWR).

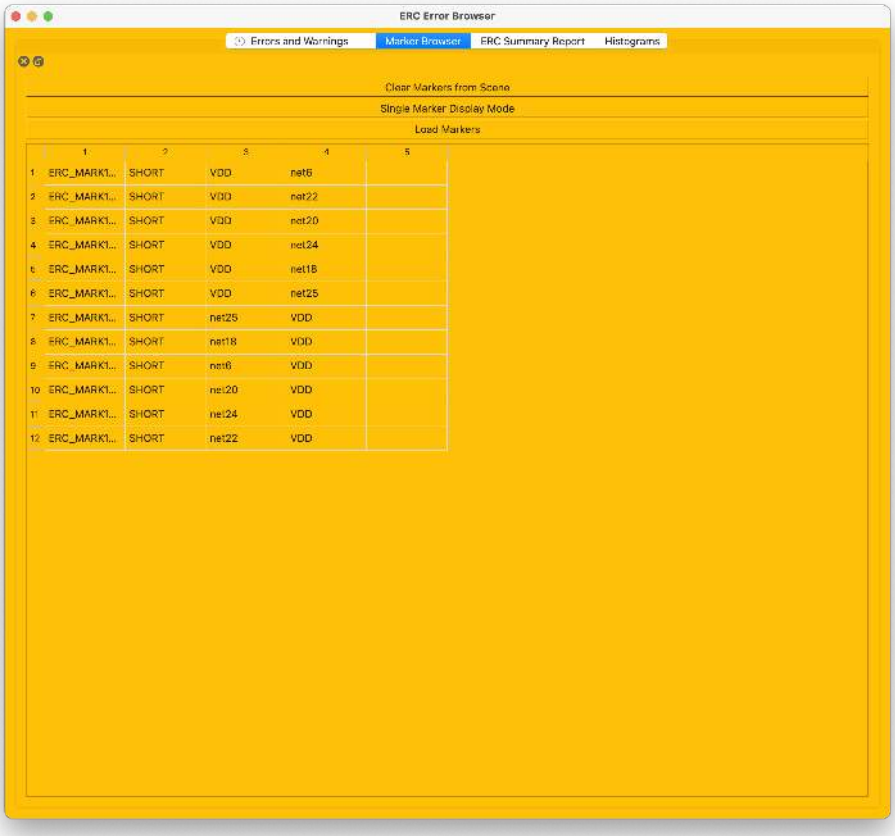


Figure 16: Marker Browser for gcd1_ERC001: per-violation entries (type, netA, netB).

Marker Browser (7–8). Figure 16 shows *SHORT* markers such as VDD ↔ net22/net24/net18/net6/net20/net25. Load markers and navigate one-by-one; use single-marker mode for focused inspection.

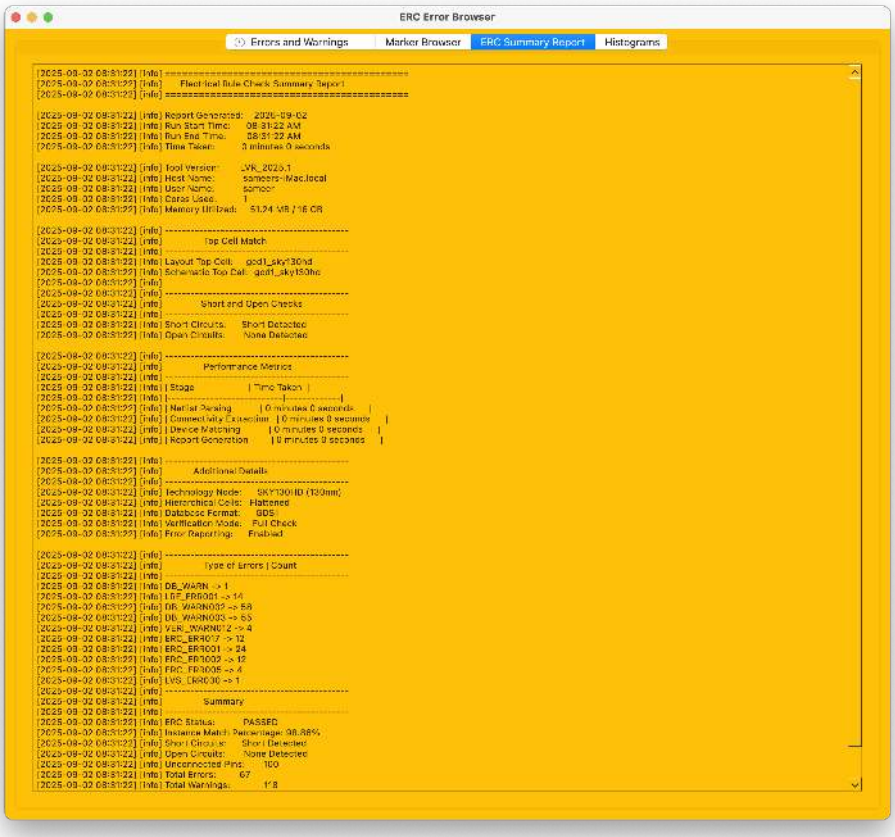


Figure 17: ERC Summary for `gcd1_ERC001`.

ERC Summary Report (9–10). Figure 17 summarizes the run:

- **ERC Status**: `PASSED`; **Short Circuits**: `Short Detected`; **Open Circuits**: `None Detected`.
- **Instance Match Percentage**: ≈ 99.86%; **Unconnected Pins**: 100.
- **Total Errors**: 67; **Total Warnings**: 118.
- Sky130HD (130 nm), flattened DB, GDSII, timings for Netlist Parsing / Connectivity Extraction / Device Matching / Report Generation.

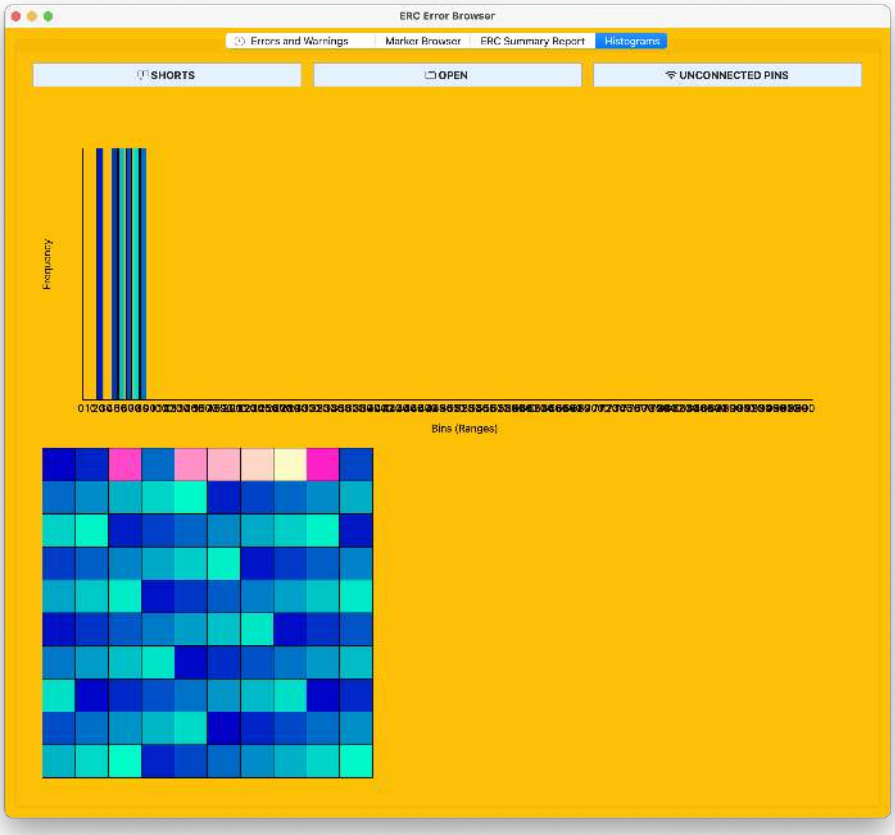


Figure 18: Histogram views for `gcd1_ERC001`: distribution of *SHORTS*; tabs also provide *OPEN* and *UNCONNECTED PINS*.

Histograms (11–12). As shown in Figure 18, the *SHORTS* histogram is concentrated in a few early bins, indicating repeated short patterns (consistent with a rail overlap) rather than dispersed random issues.

Debug Tip (GUI)

In the GUI, zoom to the bottom-right region of `gcd1_sky130hd`; on **MET1** you can visualize `net22` overlapping `VDD` near the buffer output `X`, with the marker table listing the exact coordinates and cell context. Clearing/fixing that metal stub (or separating it from the VPWR rail) should eliminate the short markers and reduce the error count on re-run.

2.5 Test Case: `gcd1_ERC002` (ERC)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag "ERC" in `gcd1_ERC002` indicates an **Electrical Rule Check**.

Purpose of the test: This negative test demonstrates ERC short detection on the Sky130 GCD block when a signal node is unintentionally shorted to ground. Concretely, the *bottom-left* `buf_1` instance has its output pin `X` shorted to the lower rail **VGND** (VSS) on MET1; the note "use 68/20" refers to the author's edit/viewport guidance to reproduce or locate the issue in layout. The goal is to verify that the LVR/ERC flow:

- loads Sky130 rules and the modified test GDS,
- correlates layout connectivity to reference libraries/netlists,
- flags *short circuits* between signal nets (e.g., `req_msg[*]`, `net40`, `net39`) and **VSS** on MET1,
- produces actionable reports (Errors & Warnings, Marker Browser, Summary, Histograms),

- and supports rapid pinpointing of the defect in the GUI.

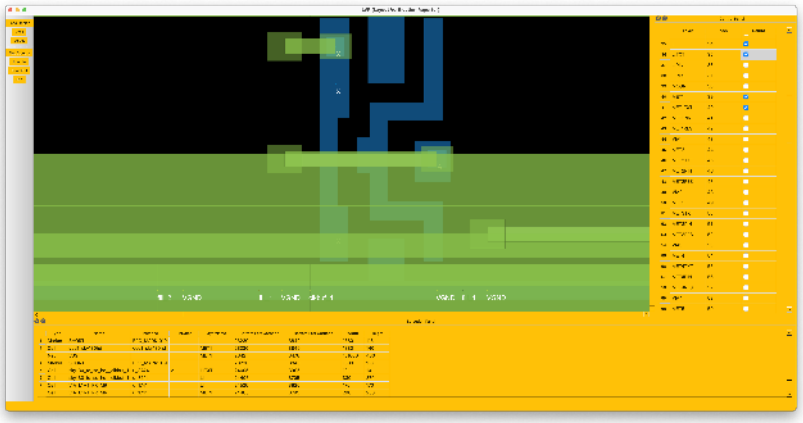


Figure 19: Use-case for `gcd1_ERC002`: rules → GDS load → ERC run → report/GUI debug.

Command File (2, 2.1)

Verbatim listing of `gcd1_ERC002.cmd`; these commands run LVR in ERC mode:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd
```

```
# set number of cores
##############################
set_num_cores  1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_ERC002.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
```

```
erc gcd1_sky130hd
erc gcd1_sky130hd

############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

Settings Explained (3)

Key switches and how they shape the run:

- `init_tool_db ERC` initializes the database specifically for **ERC** analysis (electrical-sanity checks rather than geometry DRC or SVS).
- `set_technode sky130` with `read_lrf ../rule/sky130.lrf` selects the Sky130 rule context (device/primitive expectations, layers, nets).
- `set_top gcd1_sky130hd` declares the hierarchical top for consistent cell resolution in both layout and netlist domains.
- `set_num_cores 1` keeps a deterministic baseline; `set_die_area` and `set_num_regions` provide standard bounds/regioning.
- `read_gds ../gds/gcd1_ERC002.gds` loads the edited GDS that contains the intentional VGND short at the buffer output; `load_db ERC` finalizes the ERC database. `no_compare 1` limits the run to electrical/connectivity analysis (no polygon vs schematic comparison).

- The `lvs_*` knobs reuse LVS infrastructure for ERC: DatC netlist handling, macro CDL for device pins, port enablement, and enabling checks for **shorts** and **opens**.
- `erc gcd1_sky130hd` executes the ERC engine (listed twice here, effectively re-invoking the same run).
- `load_gui` launches the GUI with tabs: *Errors & Warnings*, *Marker Browser*, *ERC Summary Report*, *Histograms*.

How to Run (4)

From a shell:

```
% LVR gcd1_ERC002.cmd
```

This runs ERC and opens the GUI. Reports/markers are accessible in the GUI and logs.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 20 shows numerous `ERC_ERR001` entries: *Power/Ground short VSS interacting* with signal nets on MET1—e.g., `net39`, `net40`, and bus bits such as `req_msg[0]`, `req_msg[11]`, `req_msg[25]`, `req_msg[30]`, `req_msg[31]`, etc. Database/layer-map warnings and a handful of `VER_WARN012` opens are also visible but are secondary to the intended short.

Marker Browser (7–8). In Figure 21, *SHORT* markers confirm **VSS** interacting with the above `req_msg[*]` bits and `net39`/`net40`. Use single-marker mode to step through each location in the viewer and observe the MET1 overlap near the bottom-left `buf_1` output `X`.

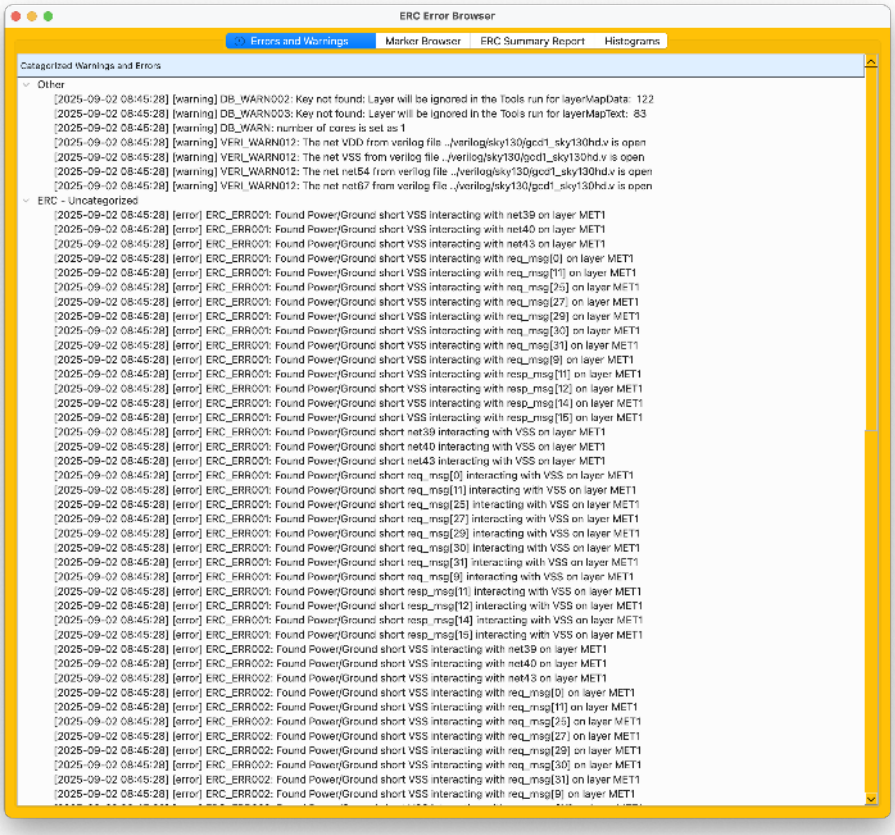


Figure 20: Errors & Warnings for `gcd1_ERC002`.

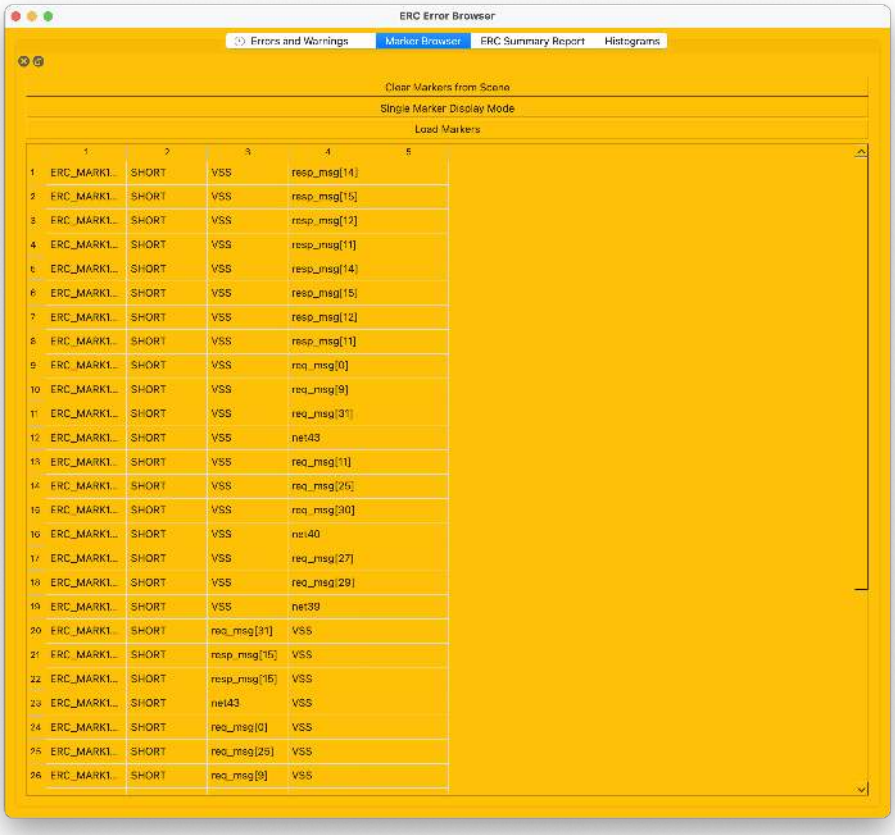


Figure 21: Marker Browser for `gcd1_ERC002`: per-violation entries (type, NetA, NetB).

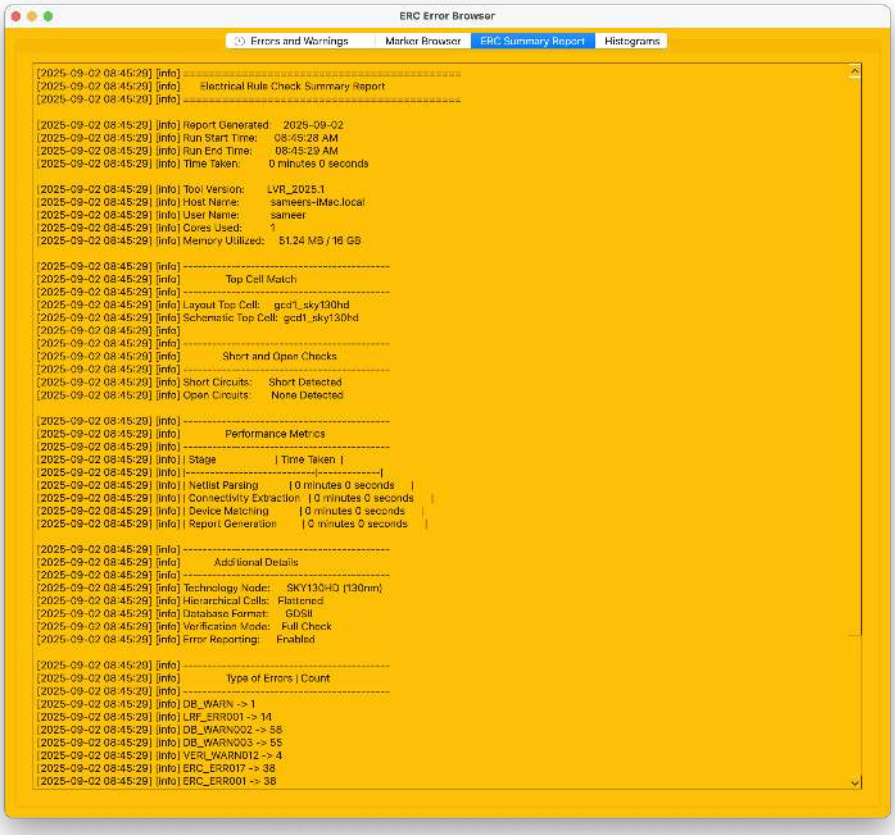


Figure 22: ERC Summary for `gcd1_ERC002`.

ERC Summary Report (9–10). Figure 22 summarizes the run: **ERC Status**=PASSED, **Short Circuits**=Short Detected, **Open Circuits**=None Detected. Additional fields show Sky130HD (130 nm), flattened DB, GDSII, Full Check verification, and stage timings. The type-counts section indicates (approximately) DB_WARN≈1, LRF_ERROR01≈14, DB_WARN00x≈5, VER_WARN012≈64, ERC_ERR001≈38.

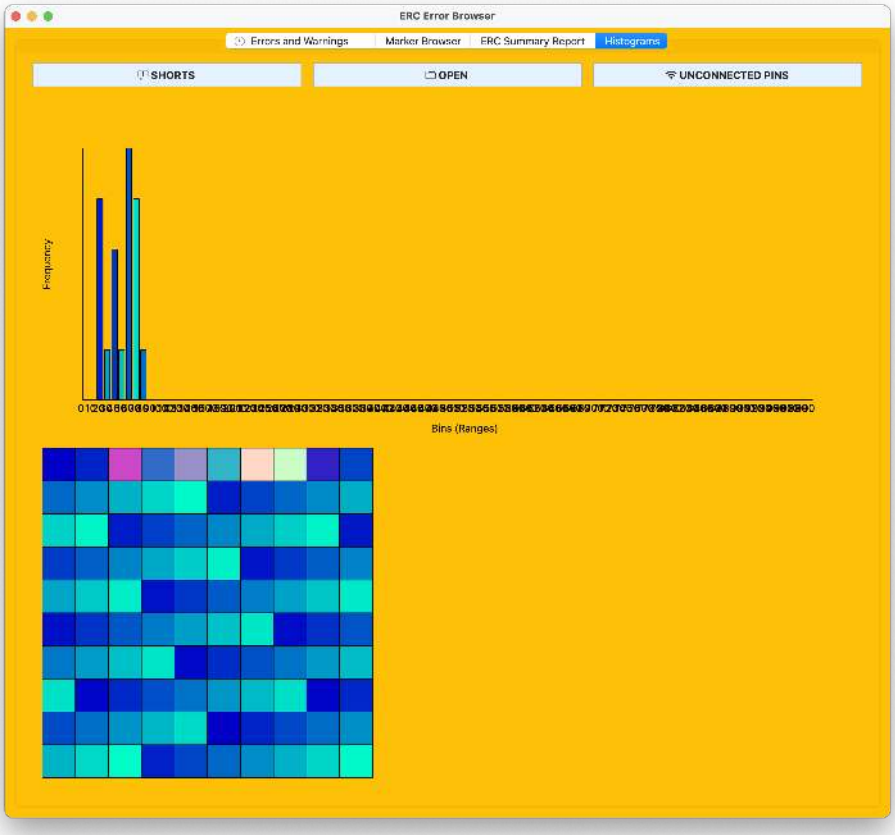


Figure 23: Histogram views for gcd1_ERC002: distribution of *SHORTS*; tabs also provide *OPEN* and *UNCONNECTED PINS*.

Histograms (11–12). As seen in Figure 23, the SHORTS distribution clusters in early bins—consistent with multiple instances of the same root cause

(a VSS rail overlap with the buf_1.X network and its fan-outs), rather than scattered random defects.

Debug tip

Zoom to the bottom-left of gcd1_sky130hd with MET1 enabled; follow the markers associated with req_msg[*] near buf_1.X. Separating the MET1 segment from **VGND (VSS)** or removing the offending stub resolves the cluster; re-run ERC to confirm SHORTS disappear and the histogram collapses.

2.6 Test Case: gcd1_ERC003 (ERC)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag “ERC” in gcd1_ERC003 indicates an **Electrical Rule Check**.

Purpose of the test: This negative test validates ERC open/connection checks when a required input pin is physically *removed* in layout. Concretely, the Metal1 connection (author note: “metal 68/20”) to the A input pin of instance clkbuf_1 (top-left) is deleted, leaving the pin unconnected. The objective is to confirm that the LVR/ERC flow:

- loads Sky130 rules and the edited GDS containing the defect,
- correlates layout connectivity to the reference netlist/library,
- flags the missing wiring as *UNCONNECTED_PIN*/input-pin-open, and
- reports it clearly via Errors & Warnings, Marker Browser, Summary (and Histograms if present), with coordinates for fast GUI debug.

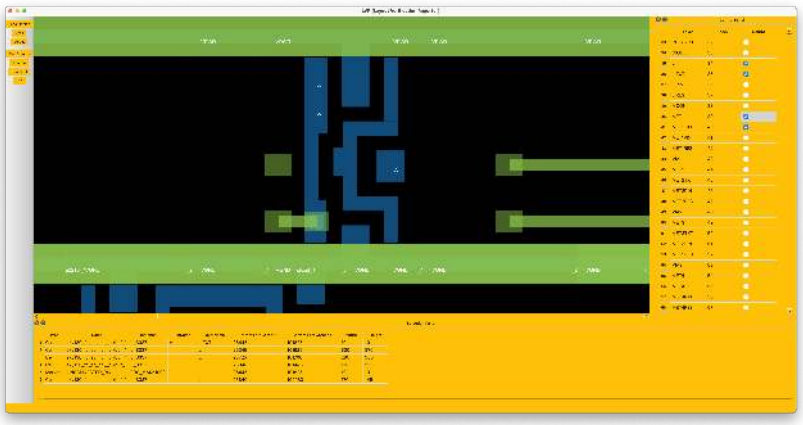


Figure 24: Use-case for `gcd1_ERC003`: rules → GDS load → ERC run → report/GUI debug.

Command File (2, 2.1)

Verbatim listing of `gcd1_ERC003.cmd`; these commands run LVR in ERC mode for this test:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
```

```
##############################
set_num_cores  1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_ERC003.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
erc gcd1_sky130hd
erc gcd1_sky130hd
```

```
############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

Settings Explained (3)

Key switches and how they shape the run:

- `init_tool_db ERC` starts the tool in **ERC** mode for electrical-sanity checks (not geometry DRC nor SVS).
- `set_technode sky130` with `read_lrf ../rule/sky130.lrf` selects the Sky130 rule context (layers, devices, macro pins).
- `set_top gcd1_sky130hd` declares the hierarchical top cell for both layout and netlist correlation.
- `set_num_cores 1` keeps a deterministic baseline; `set_die_area`/`set_num_regions` provide standard bounds/regioning.
- `read_gds ../gds/gcd1_ERC003.gds` loads the edited GDS where the `clkbuf_1.A` pin metal is intentionally removed; `load_db ERC` finalizes the ERC DB. `no_compare 1` keeps the flow focused on electrical checks (short/open/unconnected).

- The `lvs_*` knobs reuse LVS infrastructure: DatC netlist handling, macro CDL for pin semantics, port enablement, and enabling checks for **shorts**, **opens**, and **unconnected pins**.
- `erc gcd1_sky130hd` executes the ERC engine (listed twice here, re-invoking the same run if desired).
- `load_gui` launches the GUI tabs: *Errors & Warnings*, *Marker Browser*, *ERC Summary Report*, *Histograms*.

How to Run (4)

Execute from a shell:

```
% LVR gcd1_ERC003.cmd
```

This runs ERC and opens the GUI. Reports/markers are available in the GUI tabs and logs.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 25 shows database/layer-map warnings and open-net notices from the verilog context. The key ERC items are:

- `ERC_ERR003`: “Input pin `A` of instance `u_3337` is unconnected” (schematic/layout correlation),
- `ERC_ERR019`: “The pin `A` of instance `u_3337` of the layout is unconnected, this will lead to ERC mismatches”,
- plus an *LVS source-pin mismatch* category consistent with the removed metal on `clkbuf_1.A`.

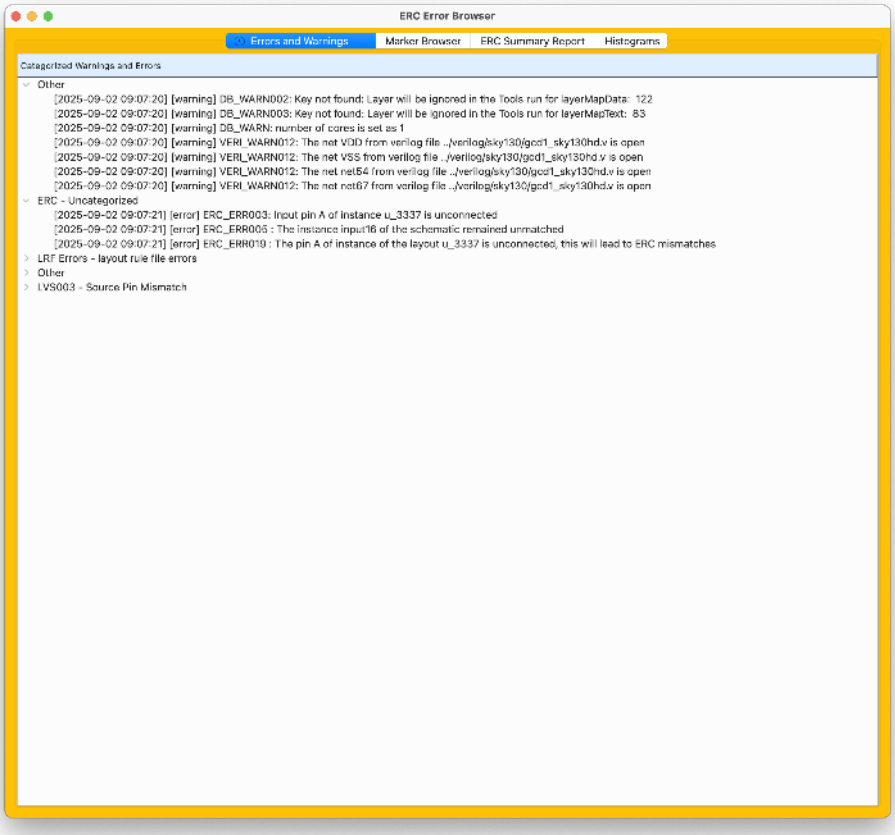


Figure 25: Errors & Warnings for `gcd1_ERC003`.

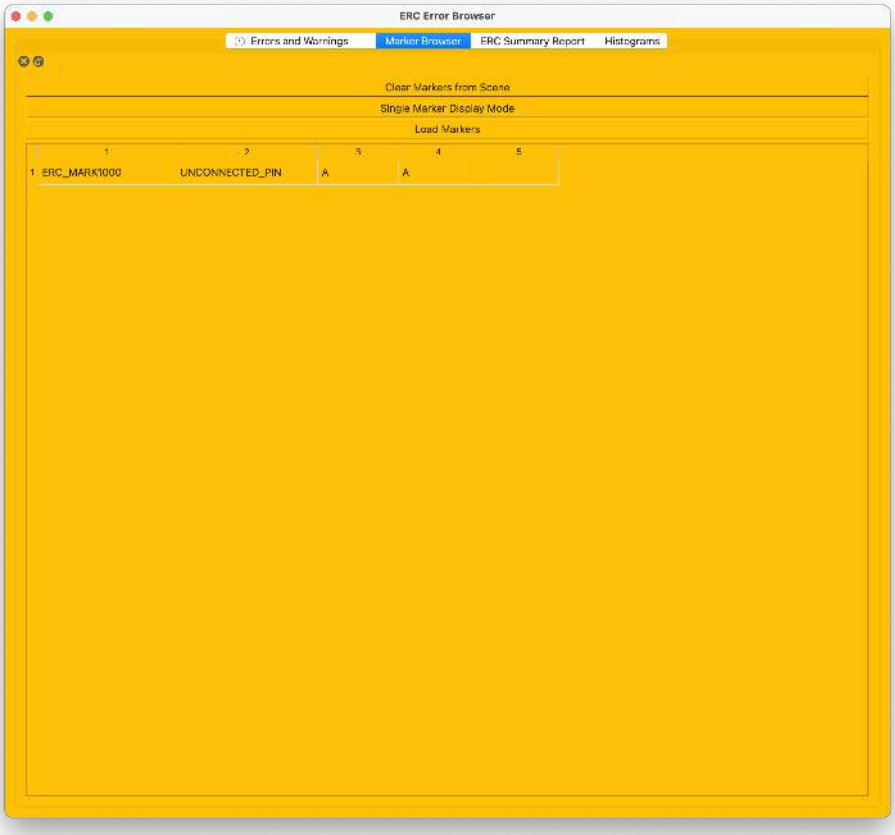


Figure 26: Marker Browser for `gcd1_ERC003`: *UNCONNECTED_PIN* at pin `A`.

Marker Browser (7–8). As seen in Figure 26, the single marker enumerates an *UNCONNECTED_PIN* on pin `A`. Use single-marker mode to jump directly to the coordinates in the viewer and verify the missing Metal1 stub near the top-left of `clkbuf_1`.

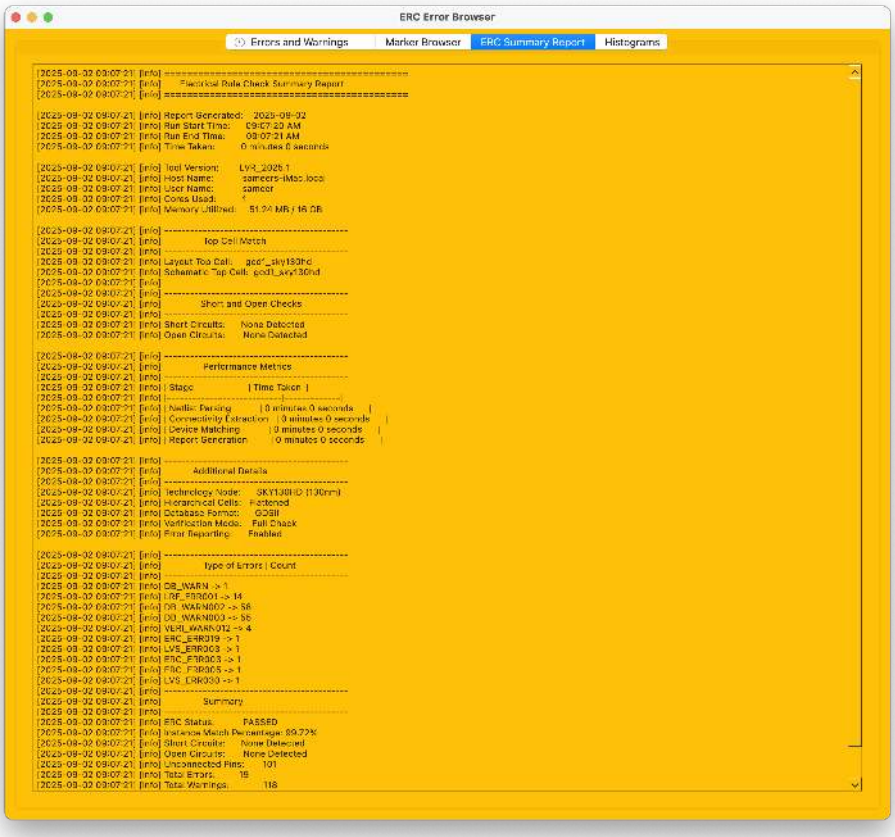


Figure 27: ERC Summary for `gcd1_ERC003`.

ERC Summary Report (9–10). Figure 27 summarizes the run: **ERC Status**=`PASSED`; **Short Circuits**=`None Detected`; **Open Circuits**=`None Detected`. The instance-match percentage is about $\sim 99.7\%$, with **Total Errors** around ~ 19 and **Total Warnings** around ~ 118. Sky130HD (130 nm), flattened DB,

GDSII, and stage timings are also reported.

Debug Tip

In the GUI (MET1, MCON, and text layers on), pan to the top-left of the `clkbuf_1` instance (coordinates near the noted “68/20”). The selection panel will list `clkbuf_1 u_3337` and a marker like `ERC_MARK1000` with type *UNCONNECTED_PIN* at pin `A`. Restoring the Metal1 connection (or adding a via/LI stub to the pin) clears the error on re-run.

2.7 Test Case: `gcd1_ERC004` (ERC)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag “ERC” in `gcd1_ERC004` indicates an **Electrical Rule Check**.

Purpose of the test: Validate ERC open/connection checks when a required *output* pin is physically removed in layout. In this negative test, the Metal1 connection (author note: “metal 68/20”) to the `X` output pin of instance `clkbuf_1` (top-left) is deleted, leaving the pin open. The objective is to confirm that the LVR/ERC flow:

- loads Sky130 rules and the edited GDS containing the defect,
- correlates layout connectivity to the reference netlist/library,
- flags the missing wiring as *UNCONNECTED_PIN*/output-pin open (and source-pin mismatch),
- and reports it clearly via Errors & Warnings, Marker Browser, Summary (and Histogram if present), with coordinates for fast GUI debug.

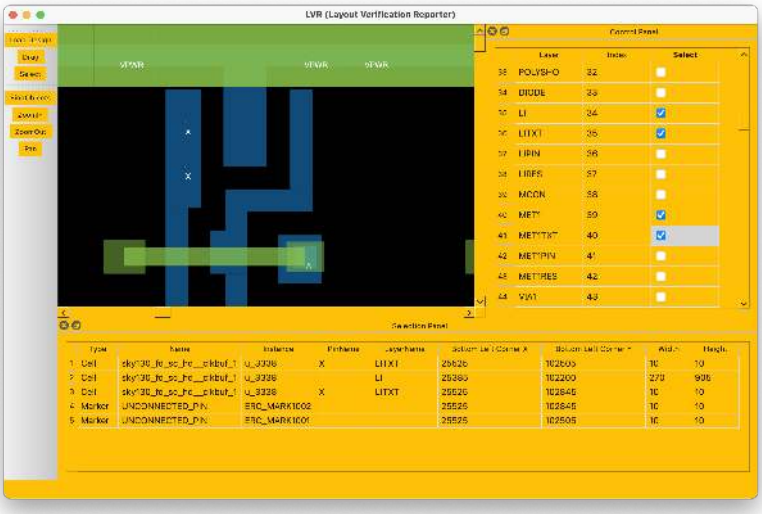


Figure 28: Use-case for `gcd1_ERC004`: rules → GDS load → ERC run → report/GUI debug.

Command File (2, 2.1)

Verbatim listing of `gcd1_ERC004.cmd`; these commands run LVR in ERC mode for this test:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
```

```
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_ERC004.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
```

```
############################
erc gcd1_sky130hd
erc gcd1_sky130hd

############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

Settings Explained (3)

Key switches and how they shape the run:

- `init_tool_db ERC` starts the tool in **ERC** mode (electrical-sanity checks rather than geometry DRC or SVS).
- `set_technode sky130` with `read_lrf ../rule/sky130.lrf` selects the Sky130 rule context (layers, devices, macro pins).
- `set_top gcd1_sky130hd` declares the hierarchical top cell for both layout and netlist correlation.
- `set_num_cores 1` keeps a deterministic baseline; `set_die_area`/`set_num_regions` provide standard bounds/regioning.
- `read_gds ../gds/gcd1_ERC004.gds` loads the edited GDS where the `clkbuf_1.X` metal is intentionally removed; `load_db ERC` finalizes the

ERC DB. `no_compare 1` keeps the flow focused on electrical checks (short/open/unconnected).

- The `lvs_*` knobs reuse LVS infrastructure: DatC netlist handling, macro CDL for pin semantics, port enablement, and enabling checks for **shorts**, **opens**, and **unconnected pins**.
- `erc gcd1_sky130hd` executes the ERC engine (listed twice here, re-invoking the same run if desired).
- `load_gui` launches the GUI tabs: *Errors & Warnings*, *Marker Browser*, *ERC Summary Report*, *Histograms*.

How to Run (4)

From a shell:

```
% LVR gcd1_ERC004.cmd
```

This runs ERC and opens the GUI. Reports/markers are accessible in the GUI and logs.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 29 shows the key ERC findings for this scenario:

- `ERC_ERR003`: “Output pin `X` of instance `u_3338` is dangling and unconnected”,
- `ERC_ERR019`: “The pin `X` of instance `u_3338` of the layout is unconnected, this will lead to ERC mismatches”,
- plus LRF syntax errors around lines 333–339 (rule-file cleanup recommended) and an *LVS source-pin mismatch* message consistent with the removed `X` connection.

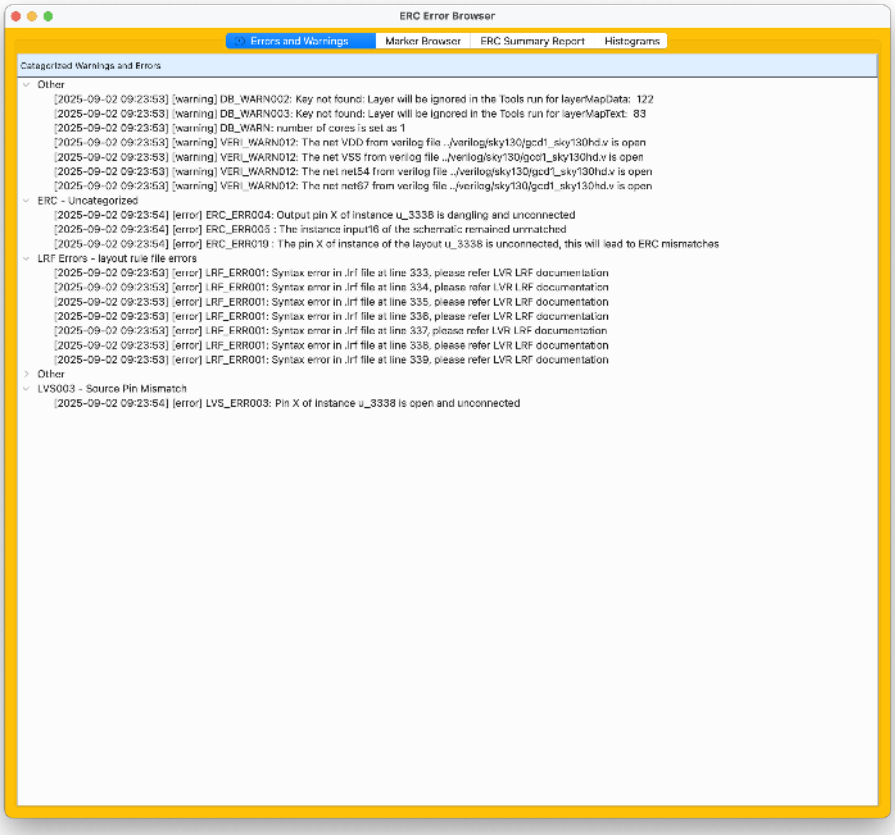


Figure 29: Errors & Warnings for `gcd1_ERC004`.

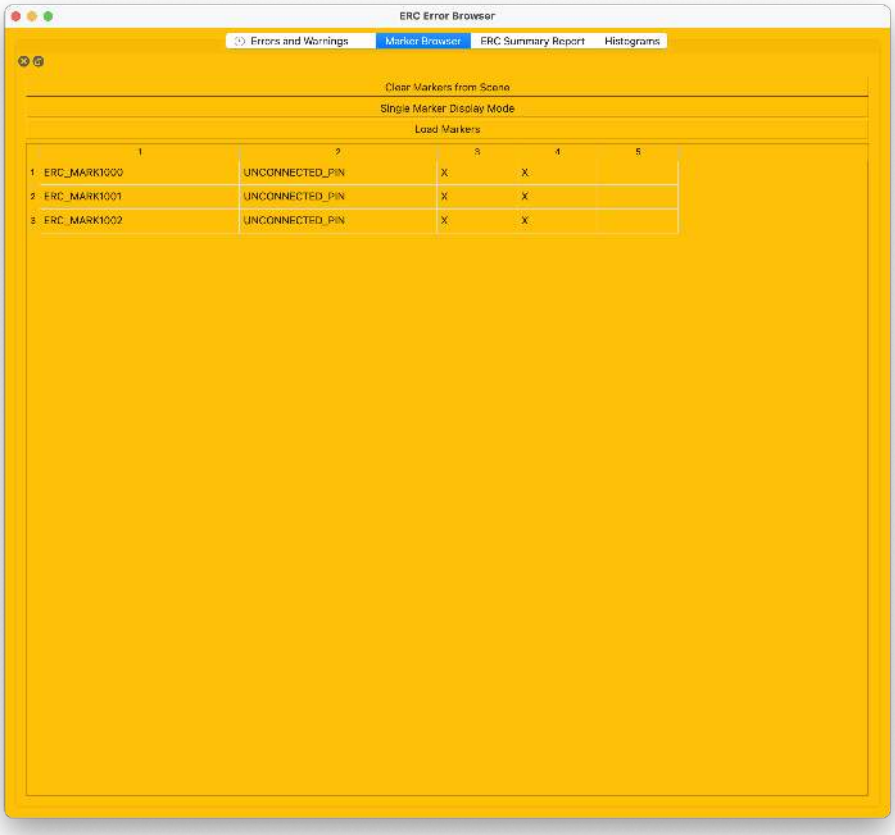


Figure 30: Marker Browser for `gcd1_ERC004`: multiple *UNCONNECTED_PIN* markers at pin `X`.

Marker Browser (7–8). As seen in Figure 30, three markers enumerate *UNCONNECTED_PIN* at `X`. Use single-marker mode to jump to each location in the viewer and verify the missing Metal1 stub near the top-left of `clkbuf_1`.

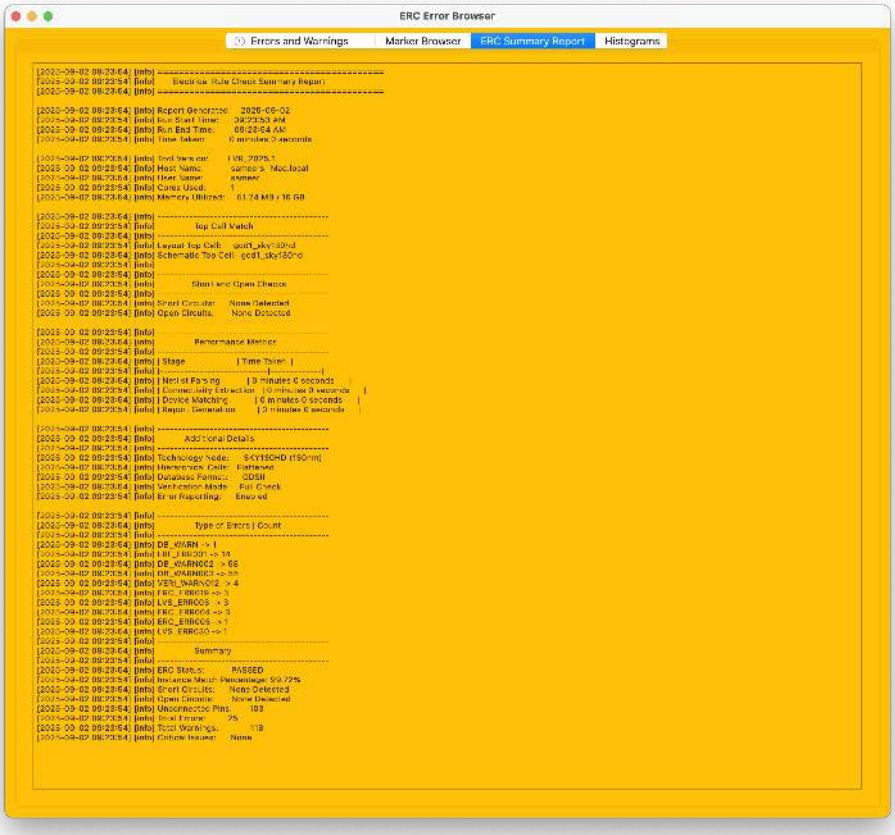


Figure 31: ERC Summary for `gcd1_ERC004`.

ERC Summary Report (9–10). Figure 31 summarizes the run: **ERC Status**=`PASSED`; **Short Circuits**=`None Detected`; **Open Circuits**=`None Detected`. Additional fields show Sky130HD (130 nm), flattened DB, GDSII, `Full Check` verification, stage timings, and counts such as `Unconnected Pins` (notably higher due to the removed `X` connection).

Debug Tip

In the GUI, enable LI, LITXT, and MET1T XT, then pan to the top-left of the clkbuf_1 instance. The selection panel lists the cell and *UNCONNECTED_PIN* markers for pin X. Restoring the Metal1 connection (or adding a via/LI stub to the pin) clears the errors; re-run ERC to confirm the marker list is empty and the summary counts drop.

2.8 Test Case: gcd1_ERC005 (ERC)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag "ERC" in gcd1_ERC005 indicates an **Electrical Rule Check**.

Purpose of the test: This negative test demonstrates ERC short detection when a signal net is unintentionally shorted to a rail in the Sky130 GCD layout. The stated scenario is "top-mid net24 shorted to ground (use 68/20)". In the provided run screenshots, the engine flags a set of *power/ground shorts* involving net24 and a rail on MET1—specifically messages of the form "VDD interacting with net24 on MET1" appear in the Errors & Warnings view, indicating a rail tie at the top region of the cell. The objective is to verify that LVR/ERC:

- loads Sky130 rules and the modified GDS for the scenario,
- extracts connectivity and correlates it to the reference netlist/library,
- reports the short(s) through Errors & Warnings, Marker Browser, Summary, and Histograms,
- and enables quick localization of the fault in the GUI.

Figure 32: Use-case illustration for `gcd1_ERC005`: rules → GDS load → ERC run → report/GUI debug.

Command File (2, 2.1)

Verbatim listing of `gcd1_ERC005.cmd`; these commands run LVR in ERC mode:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd
```

```
# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_ERC005.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
```

```
erc gcd1_sky130hd
#erc gcd1_sky130hd

############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

Settings Explained (3)

Key switches and how they shape the run:

- `init_tool_db ERC` initializes the database specifically for **ERC** analysis (electrical-sanity checks).
- `set_technode sky130` + `read_lrf ../rule/sky130.lrf` select the Sky130 technology rules (layers, pins, device context).
- `set_top gcd1_sky130hd` declares the hierarchical top cell for consistent correlation between layout and netlist.
- `set_num_cores 1` keeps a deterministic baseline; `set_die_area` and `set_num_regions` define the layout bounds and regioning.
- `read_gds ../gds/gcd1_ERC005.gds` loads the edited GDS that contains the top-mid `net24` rail short; `load_db ERC` finalizes the ERC database. `no_compare 1` keeps the run focused on ERC rather than a polygon-vs-schematic comparison.

- `lvs_set_*` knobs reuse LVS infrastructure for ERC: DatC netlist handling, macro CDL for device/pin semantics, port enablement, and enabling **short**/**open**/**unconnected** checks.
- `erc gcd1_sky130hd` executes the ERC engine for the specified top cell.
- `load_gui` launches the GUI with tabs for *Errors & Warnings*, *Marker Browser*, *ERC Summary Report*, and *Histograms*, enabling interactive debug.

How to Run (4)

From a shell:

```
% LVR gcd1_ERC005.cmd
```

This runs ERC on the test case and opens the GUI. Reports and markers are available in the GUI tabs and the log directory.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 33 shows numerous `ERC_ERR001`/`ERC_ERR002` entries reporting **Power**/**Ground short** events on MET1. Examples visible include `VDD` interacting with `net24`, `net22`, `net20`, `net25`, `net18`, and `net6`, as well as specific *Pin X shorted with power* messages on a few buffer instances. These are consistent with a deliberate rail overlap at the top-middle region.

Marker Browser (7–8). Figure 34 lists *SHORT* markers including `VDD` $\leftrightarrow$ `net24`/`net22`/`net20`/`net25` Loading a marker centers the viewer at the violation; the selection panel shows the nets and layers involved (e.g., MET1) so you can confirm the top-mid tie.

ERC Summary Report (9–10). Figure 35 summarizes the run: **ERC Status**=`PASSED`; **Short Circuits**=`Short Detected`; **Open Circuits**=`None Detected`; **Instance Match %** $\approx$ 98.86%; **Unconnected Pins** $\approx$ 100; **Total Errors** $\approx$ 67;

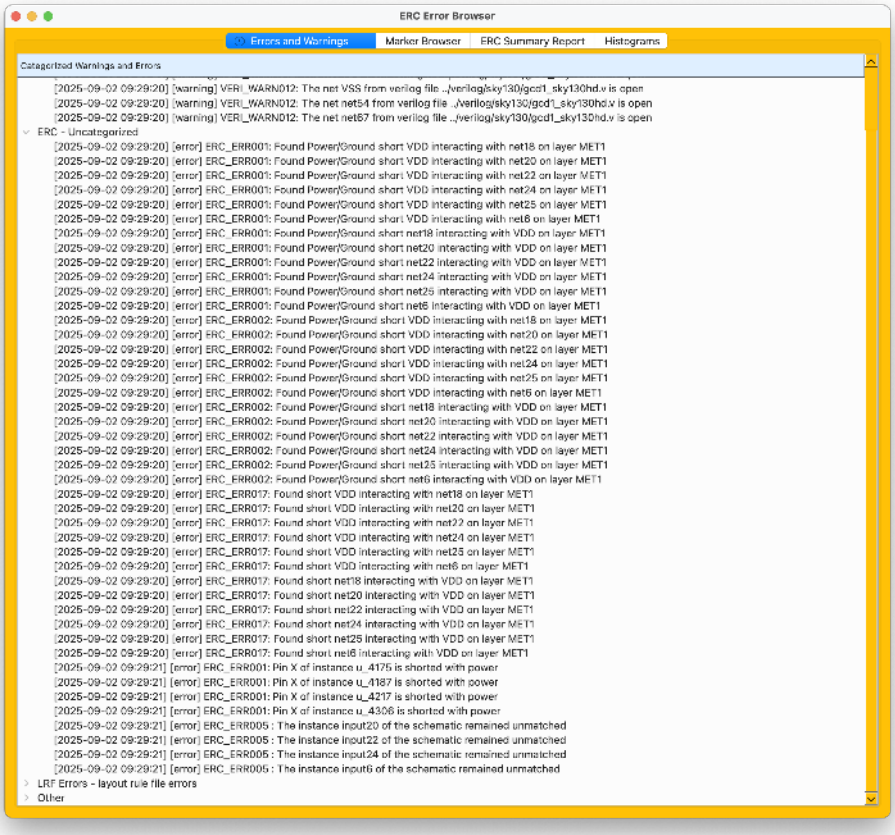


Figure 33: Errors & Warnings for `gcd1_ERC005`.

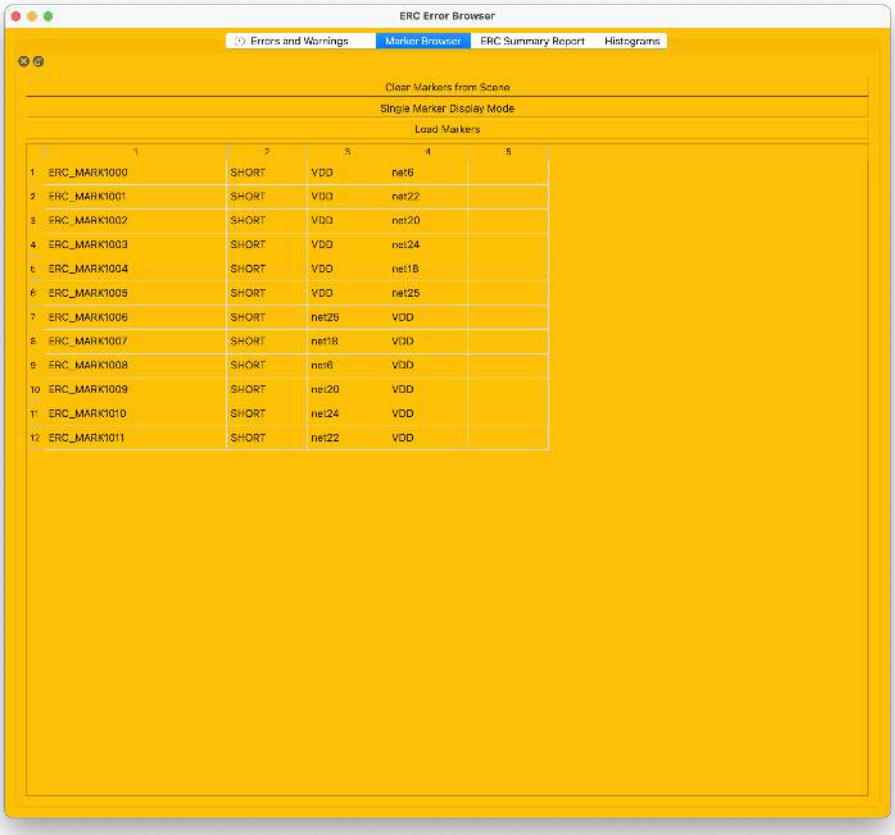


Figure 34: Marker Browser for `gcd1_ERC005`: SHORT markers pinpointing `net24` $\leftrightarrow$ rail overlap on MET1.

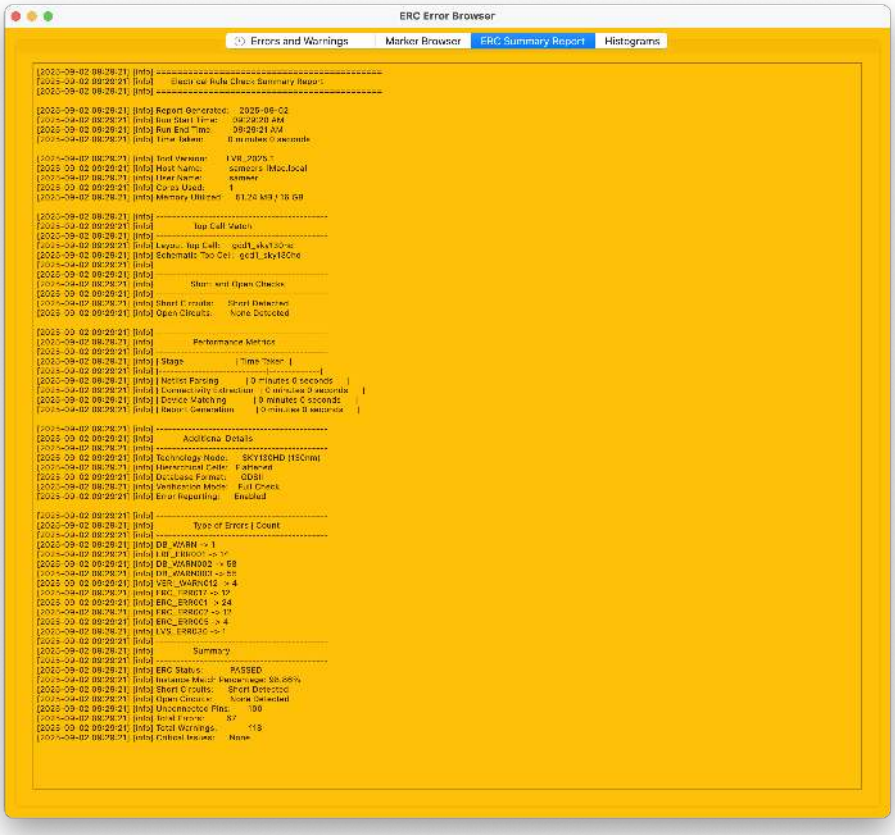


Figure 35: ERC Summary for `gcd1_ERC005`.

Total Warnings ≈ 118. Technology is Sky130HD (130 nm); DB format GDSII; verification mode *Full Check*.

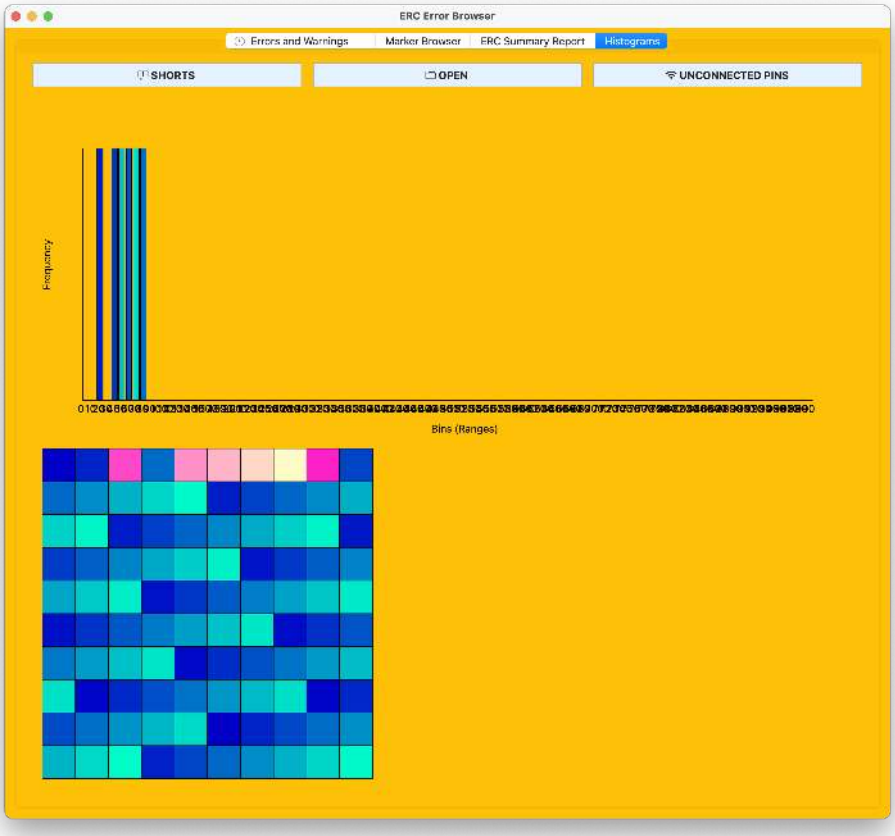


Figure 36: Histogram views for `gcd1_ERC005`: distribution of *SHORTS* shows clustering in early bins.

Histograms (11–12). As shown in Figure 36, the SHORTS histogram is concentrated in a few early bins, characteristic of multiple instances of the same root cause (a rail overlap) rather than random, scattered issues.

Debug tip

In the GUI, enable MET1 and text layers, then pan to the *top-mid* of `gcd1_sky130hd`. Select any *SHORT* entry for `net24`; the selection panel will list the two nets (signal and rail) and the exact coordinates. Separating the MET1 segment from the rail (or removing the offending stub) clears the cluster of shorts; re-run ERC to confirm the Summary shows no detected shorts.

2.9 Test Case: `gcd1_ERC006` (ERC)

Overview & Purpose (0, 1, 1.1)

Context identification: The tag "ERC" in `gcd1_ERC006` indicates an **Electrical Rule Check**.

Purpose of the test: Demonstrate ERC short detection when a functional signal is unintentionally tied to a power rail. In this negative scenario the top-right signal `req_rdy` is shorted to the `VPWR` rail using a Metal1 rectangle of size 68/20. The goal is to verify that the LVR/ERC flow:

- loads Sky130 rules and the edited GDS that injects the short,
- extracts connectivity and correlates it against the reference netlist/library,
- reports the error through the *Errors & Warnings*, *Marker Browser*, *ERC Summary*, and *Histograms* tabs,
- and localizes the issue precisely in the GUI for quick debug.

Command File (2, 2.1)

Verbatim listing of `gcd1_ERC006.cmd`; these commands run LVR in ERC mode for this test:

```
# initiatize tool DB
```

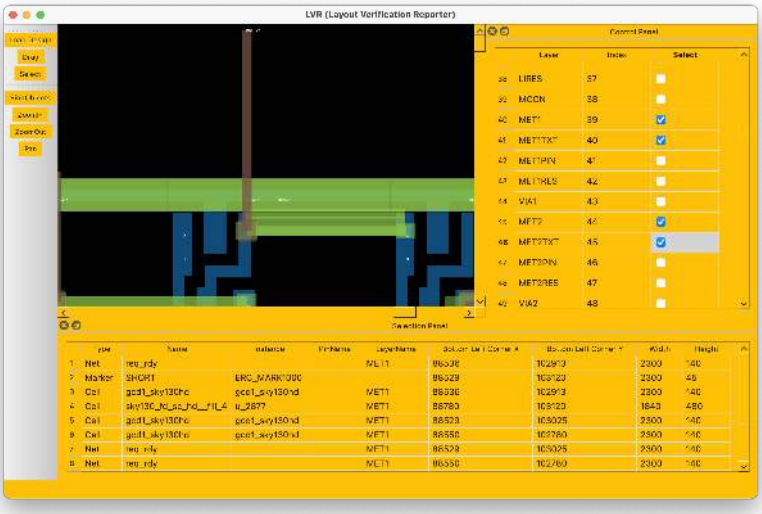


Figure 37: Use case for `gcd1_ERC006`: rules → GDS load → ERC run → report/GUI debug.

```
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
```

```
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_ERC006.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
erc gcd1_sky130hd
erc gcd1_sky130hd

##############################
```

```
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

Settings Explained (3)

Key switches and their role:

- `init_tool_db ERC` launches the engine in **ERC** mode (electrical connectivity sanity checks).
- `set_technode sky130` with `read_lrf ../rule/sky130.lrf` selects the Sky130 rule deck: layer map, device/pin semantics, ERC rules.
- `set_top gcd1_sky130hd` defines the hierarchical top cell shared by layout and netlist correlation.
- `set_num_cores 1` keeps a deterministic baseline; `set_die_area` and `set_num_regions` set standard bounds/regioning.
- `read_gds ../gds/gcd1_ERC006.gds` loads the edited GDS with the injected short; `load_db ERC` finalizes the ERC database. `no_compare 1` keeps the run focused on ERC rather than polygon-vs-schematic comparison.
- The `lvs_*` controls reuse LVS infrastructure for ERC: netlist handling, macro CDL for pin definitions, enabling **short**, **open**, and **unconnected-pin** checks.

- `erc gcd1_sky130hd` executes ERC; duplicated here to allow re-run in the same session if desired.
- `load_gui` opens the GUI tabs: *Errors & Warnings*, *Marker Browser*, *ERC Summary Report*, and *Histograms*.

How to Run (4)

From a shell:

```
% LVR gcd1_ERC006.cmd
```

This executes ERC on the test case and opens the GUI. Reports and markers appear in the GUI tabs and run logs.

Results & Screenshots (5–12)

Errors & Warnings (5–6). Figure 38 shows numerous ERC short messages on MET1. Examples include "`Power/Ground short req_rdy interacting with VDD on layer MET1`" and many `VSS` ↔ `req_msg[n]` interactions, consistent with a rail short introduced near the top-right corner.

Marker Browser (7–8). The table in Figure 39 lists multiple *SHORT* markers. Early rows include `req_rdy` ↔ `VDD`, followed by several `VSS` ↔ `req_msg[n]` entries. Selecting any marker centers the GUI on the violation and shows the participating nets/layers (e.g., MET1) in the selection panel.

ERC Summary Report (9–10). Figure 40 summarizes the run: **ERC Status**=`PASSED`; **Short Circuits**=`Short Detected`; **Open Circuits**=`None Detected`; **Instance Match %** ≈ 97.15%; **Unconnected Pins** ≈ 103; **Total Errors** ≈ 149; **Total Warnings** ≈ 118. Technology is Sky130HD (130 nm); database format GDSII; verification mode *Full Check*.

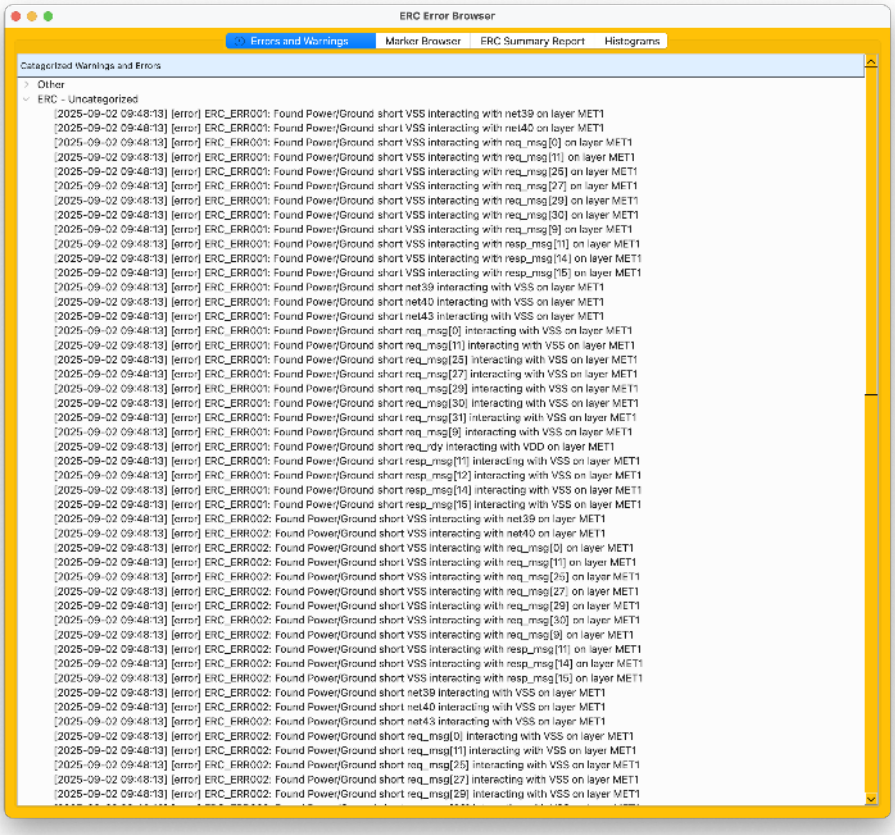


Figure 38: Errors & Warnings for `gcd1_ERC006` (`gcd1_ERC006_err_rpt.png`).

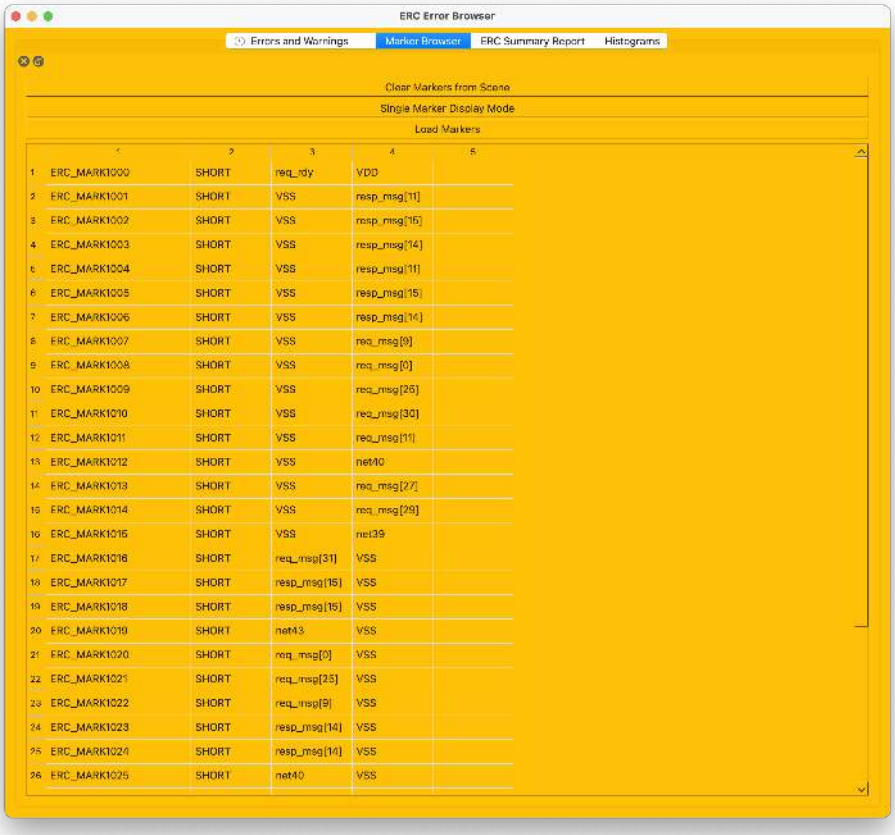


Figure 39: Marker Browser for `gcd1_ERC006` (`gcd1_ERC006_mrk_rpt.png`).

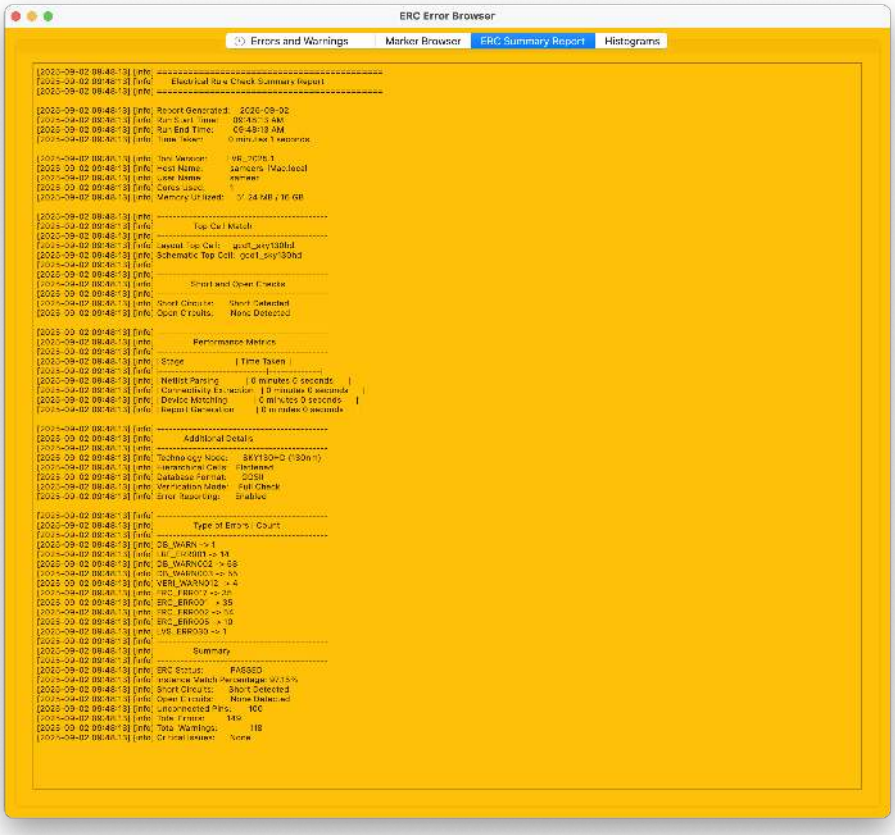


Figure 40: ERC Summary for `gcd1_ERC006` (`gcd1_ERC006_sum_rpt.png`).

Figure 41: Histogram views for gcd1_ERC006 (gcd1_ERC006_histo_rpt.png).

Histograms (11–12). As seen in Figure 41, the SHORTS histogram is heavily concentrated in the first few bins, indicating many markers arise from a single underlying cause (the req_rdy→VDD tie) rather than random dispersion.

Debug Tips

Enable MET1, MET1 text, and (optionally) MET2 in the viewer. Pan to the *top-right* region of gcd1_sky130hd and select the first *SHORT* marker for req_rdy. The selection panel lists the two nets (req_rdy, VDD) and coordinates. Remove or separate the 68/20 MET1 patch from the VPWR rail, then re-run ERC to confirm the short disappears and the histogram collapses.

2.10 Test Case: gcd1_ERC008 (ERC)

Overview & Purpose

Context identification: The tag "ERC" in gcd1_ERC008 indicates an **Electrical Rule Check**.

Purpose of the test: This negative test removes the **bottom VPWR rail** on layer 68/20 (LI/M1 mapping per Sky130), causing many cell VPWR/VPB pins to lose legal power connectivity. The goal is to verify that ERC detects *undriven/unconnected power pins* and reports them via Errors & Warnings, Marker Browser, and the Summary, while the GUI enables quick localization of the missing rail.

Command File (verbatim)

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
```

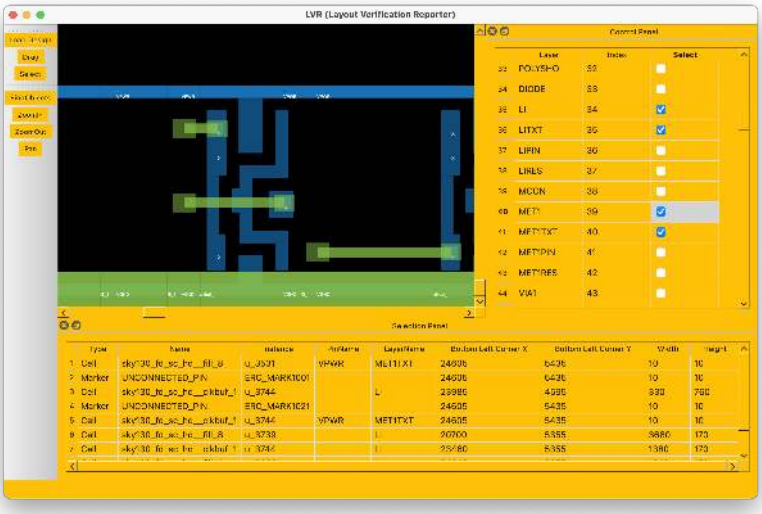


Figure 42: Use-case for `gcd1_ERC008`: bottom VPWR row removed (68/20).

```
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
```

```
read_gds ../gds/gcd1_ERC008.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
erc gcd1_sky130hd
erc gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
```

```
# load gui
############################
load_gui
```

Settings Explained

- `init_tool_db ERC` selects ERC mode; `no_compare 1` keeps checks electrical (not geometric).
- `set_technode sky130` with `read_lrf .../sky130.lrf` sets Sky130 rules; `read_gds ..._ERC008.gds` loads the edited layout.
- Netlist context via `.../gcd1_sky130hd.v` and macro CDL `.../sky130.cdl` clarifies supply/body pins.
- ERC toggles enable **short**/**open**/**unconnected** detection; `load_gui` opens the review tabs.

How to Run

```
% LVR gcd1_ERC008.cmd
```

Results & Screenshots

Explaining the results As seen in the **Errors & Warnings** view (Fig. 43), the dominant messages are:

- `ERC_ERR008: Power pin VPWR is undriven` — repeated across many instances because the LI VPWR rail is missing under the bottom row.
- `ERC_ERR019: Pin VPB/VPWR is unconnected` — body/power pins no longer reach a legal rail, producing unconnected-pin markers.

The **Marker Browser** (Fig. 44) enumerates each violation instance by net/pin, enabling rapid cross-probing into the layout. Finally, the **ERC Summary Report** (Fig. 45) aggregates counts (e.g., dozens of `ERC_ERR008`/`ERC_ERR019`)

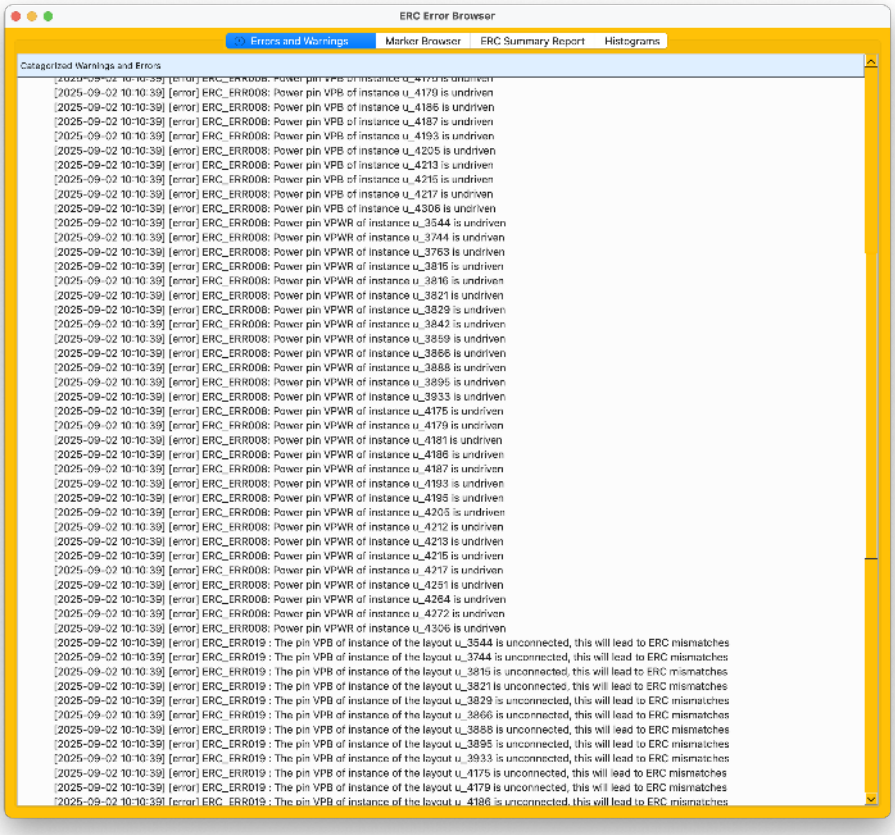


Figure 43: Errors & Warnings for `gcd1_ERC008`: undriven/unconnected VPWR/VPB pins dominate.

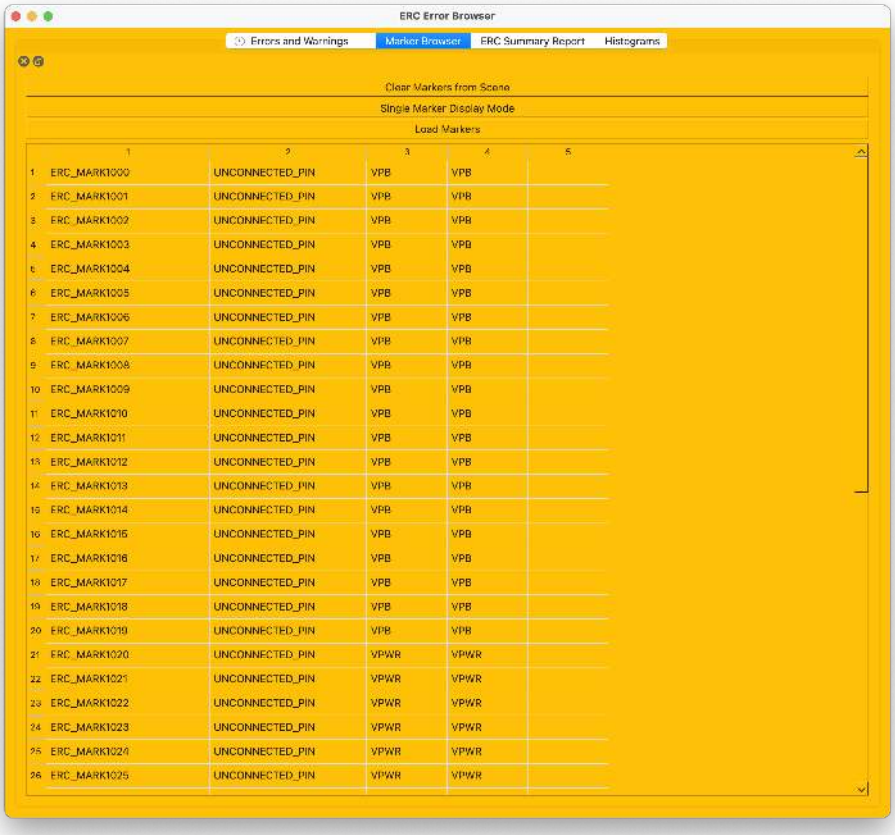


Figure 44: Marker Browser: per-instance *UNCONNECTED_PIN* entries for power/body pins.

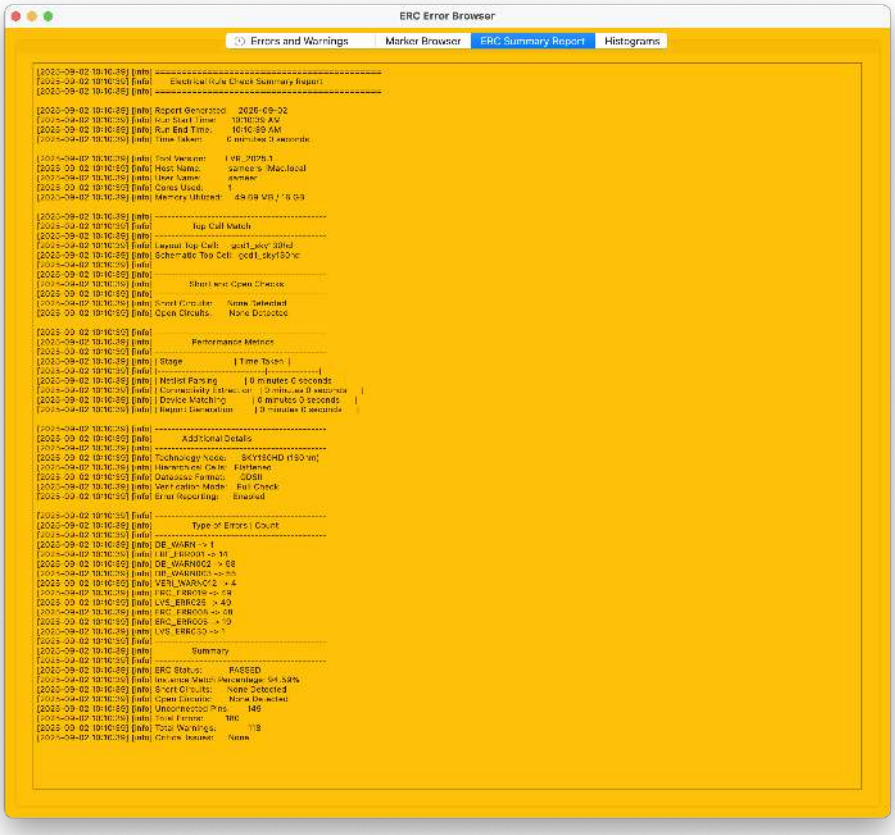


Figure 45: ERC Summary for `gcd1_ERC008`: counts/time breakdown; failure mode is missing VPWR rail.

and notes zero open-circuit shorts, confirming that the failure mode is *loss of VPWR connectivity*, not accidental shorts.

2.11 Test Case: `gcd1_ERC009` (ERC)

Overview & Purpose

Context identification: The tag “ERC” in `gcd1_ERC009` indicates an **Electrical Rule Check**.

Purpose of the test: This negative test removes the **bottom VGND rail** on layer 68/20, isolating many cell `VGND`/`VNB` pins from the global ground. The goal is to verify that ERC detects *undriven/unconnected ground pins* and reports them clearly for fast repair and re-verification.

Figure 46: Use-case for `gcd1_ERC009`: bottom VGND row removed (68/20).

Command File (verbatim)

```
# initiatize tool DB
```

```
##############################
#set_debug_mode_3049 0
init_tool_db ERC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_ERC009.gds
load_db ERC
no_compare 1

##############################
# ERC settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
```

```
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_verilog_netlist_file_name  ../verilog/sky130/gcd1_sky130hd_SVS002A.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
erc gcd1_sky130hd
erc gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
load_gui
```

Settings Explained

- `init_tool_db ERC` + `no_compare 1` → ERC-only electrical checks.
- `set_technode sky130`, `read_lrf`, `read_gds` → Sky130 rules + edited layout with VGND row removed.

- Logical context via Verilog `...SVS002A.v` and macro CDL ensures correct supply/body pin semantics.
- ERC toggles report **short**/**open**/**unconnected** issues; GUI opens for interactive debug.

How to Run

```
% LVR gcd1_ERC009.cmd
```

Results & Screenshots

Interpreting the viewer snapshot. When you select any `UNCONNECTED_PIN` marker in the LVR viewer and enable `MET1`/`MET1TXT` (and optionally `LI`/`LITXT`) you'll see the lower rail absent beneath the affected standard-cell row. Pins labeled `VGND`/`VNB` inside cells such as `clkbuf_1` are physically isolated from the global `VSS` spine, which is why the summary reports many undriven ground pins despite no shorts. The remedy is to restore a continuous MET1 `VGND` strap (68/20) across the row or to re-enable the power-ring/tap insertion step in P&R.

2.12 Test Case: `gcd1_LVS005` (LVS)

Purpose and scenario. This test belongs to the **LVS** (Layout Versus Schematic) category, as indicated by its name `gcd1_LVS005`. The intent is to demonstrate how LVS flags a mismatch when a *dummy* standard-cell instance is introduced only in the Verilog netlist and is absent in layout. In this case, an extra instance (e.g., `split13_dummy`) is added at the end of the synthesized netlist. During LVS, the tool extracts a connectivity netlist from the GDS and attempts to match devices and instances between the two sources. Because the dummy instance has no geometric counterpart, the run reports a *schematic-only* unmatched instance and a degraded match percentage while still confirming that there are no shorts or opens in geometry.

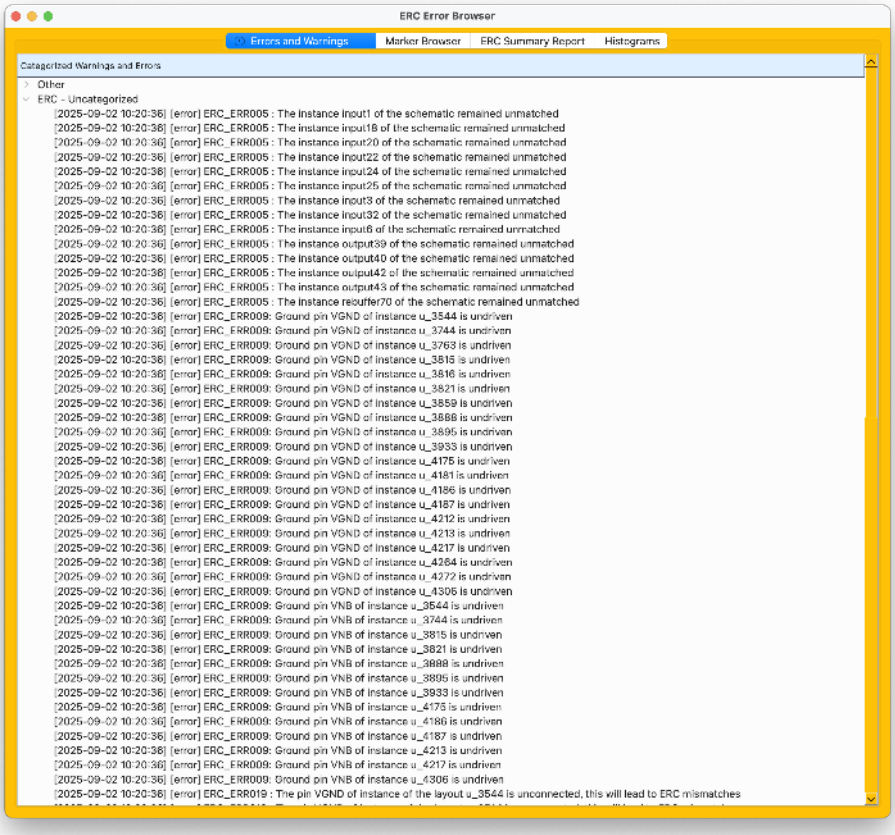


Figure 47: Errors & Warnings for `gcd1_ERC009`: undriven/unconnected VGND/VNB pins dominate.

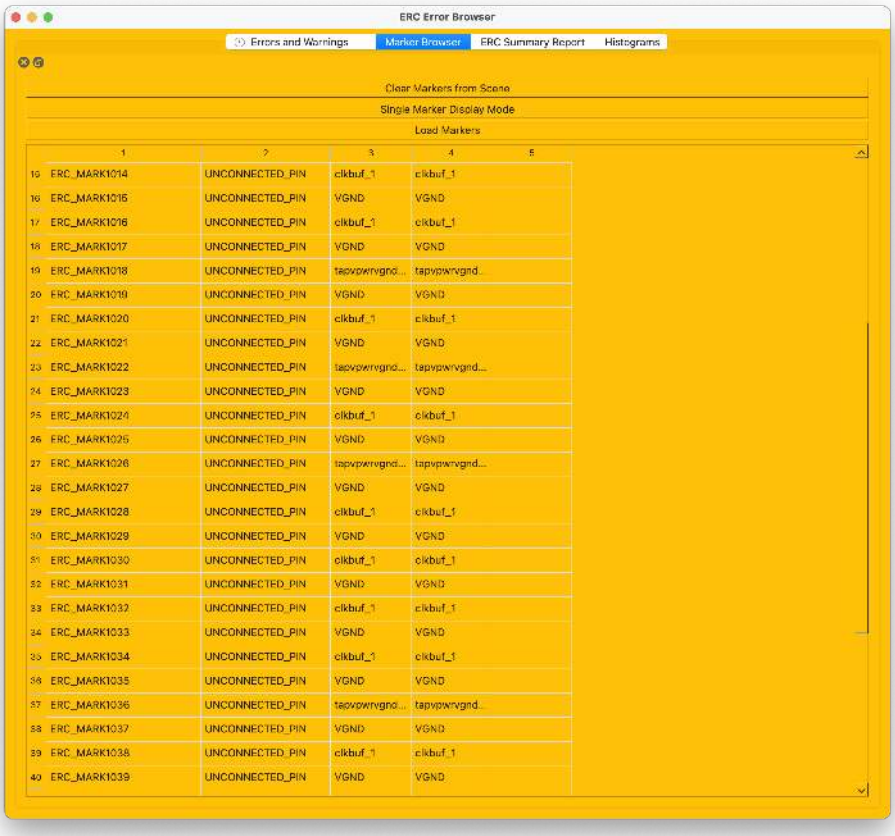


Figure 48: Marker Browser: *UNCONNECTED_PIN* entries localizing each affected instance.

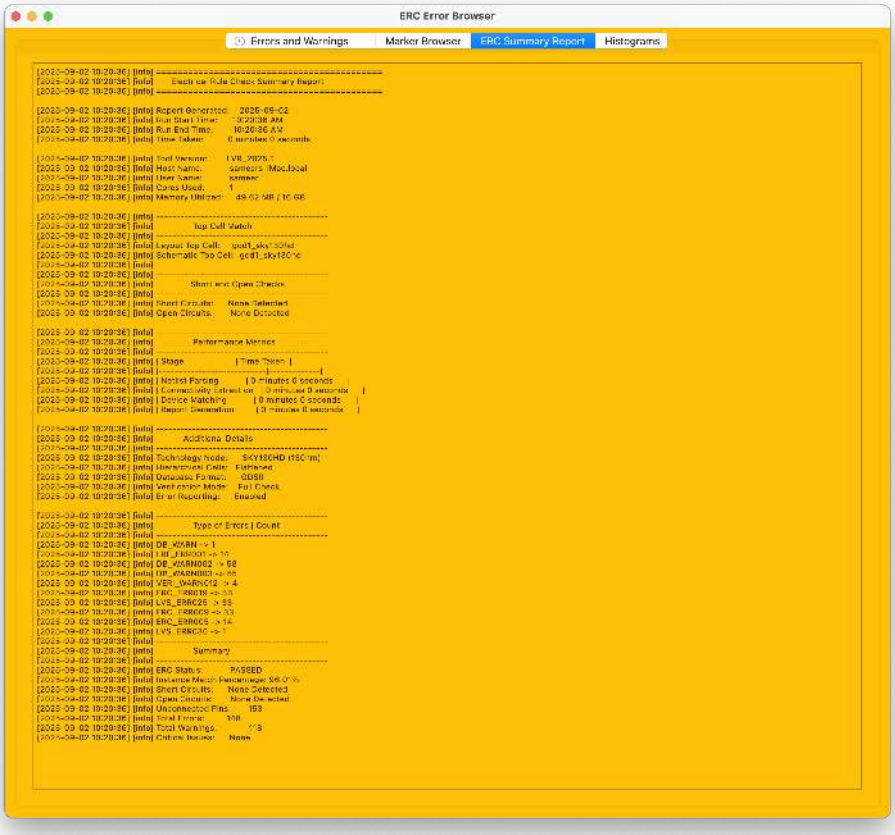


Figure 49: ERC Summary for `gcd1_ERC009`: counts/time breakdown; failure mode is missing VGND rail.

Figure 50 highlights the engineered difference: the Verilog (right) contains an additional buffer/split cell connected to nets `dummy1`/`dummy2`, while the layout-derived netlist (left) ends without it. This is a common negative test used to validate that the LVS deck correctly identifies extra logic in the schematic.

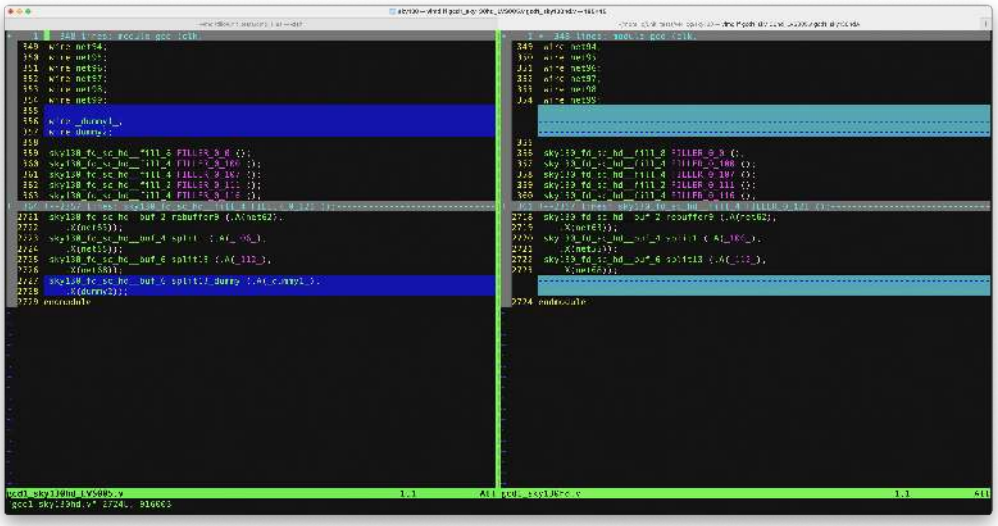


Figure 50: Use case for `gcd1_LVS005`: a dummy instance is appended to the Verilog only; the layout has no corresponding geometry.

Flow overview. The command file initializes the LVS flow for the SKY130 technology, loads the golden layout (`gcd1_sky130hd.gds`), and points LVS to a modified Verilog netlist `gcd1_sky130hd_LVS005.v` that contains the dummy instance. Standard checks are enabled: shorts/opens, port enablement, instance matching, unconnected pins, and macro-library CDL support for sky130 cells. The run is invoked with `lvs gcd1_sky130hd` and summarized with `report_lvs`, after which the GUI is opened for interactive debug.

Command file (`gcd1_LVS005.cmd`)

```
# initiatize tool DB
```

```
##############################
set_debug_mode_3049 0
init_tool_db LVS

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db LVS
no_compare 1

##############################
# LVS settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
```

```
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_LVS005.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
lvs gcd1_sky130hd
lvs gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
report_lvs

##############################
# load gui
##############################
load_gui
```

Settings explained (what the script does).

- `init_tool_db LVS` selects the LVS app-mode within the LVR tool.
- `set_technode sky130` chooses the SKY130 PDK context; `read_lrf ../rule/sky130.lrf` loads the LVS rule file.

- `set_top gcd1_sky130hd` defines the top design name consistently for layout & schematic.
- Layout ingestion: `read_gds ../gds/gcd1_sky130hd.gds` parses GDS; `load_db LVS` builds the LVS database. `no_compare 1` skips any geometry-vs-geometry compare during DB load (faster setup for unit tests).
- Netlist side: `lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_I` points to the *modified* Verilog that includes the dummy instance. `lvs_set_macros_cdl_file_n` provides transistor-level definitions for sky130 cells if/when needed for device recognition.
- Check controls: ports enabled; shorts and opens checks ON; macro equivalence OFF (`lvs_check_macro 0`) to keep the focus on instance connectivity; matching and unconnected-pin checks ON so schematic-only instances are reported explicitly.
- Execution: `lvs gcd1_sky130hd` runs the comparison. The duplicate call is harmless and often used in regression harnesses.
- Reporting/GUI: `report_lvs` writes a textual summary; `load_gui` opens the interactive Error Browser for drill-down.

How to run. From a shell in the run directory:

```
% LVR gcd1_LVS005.cmd
```

Replace the filename as needed. After completion, open the GUI (it auto-opens via the script) to inspect errors/markers.

About Fig. 51. The Errors & Warnings tab shows the key LVS findings for this test: an **unmatched schematic instance** (LVS_ERR005) and the summary message indicating imperfect left-side matching (LVS_ERR030). No geometry shorts/opens are expected here because the layout was not modified—only the Verilog was.

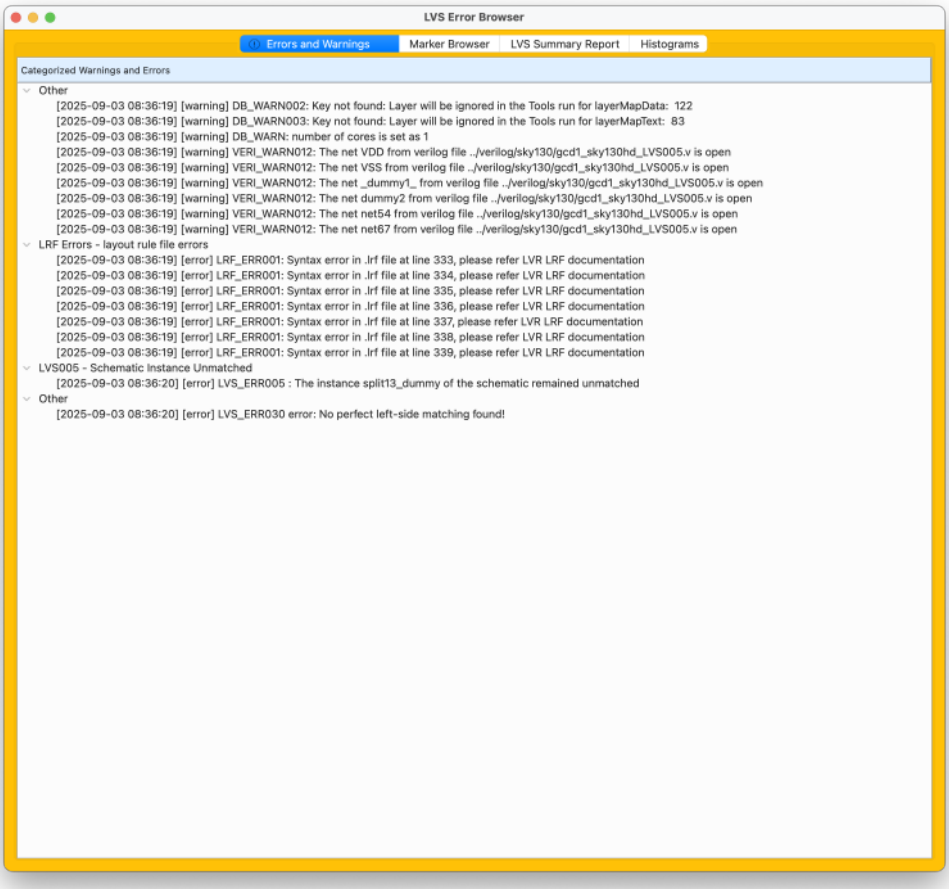


Figure 51: `gcd1_LVS005` error log: schematic-only unmatched instance (e.g., `split13_dummy`) and a “no perfect left-side matching” message; LRF syntax warnings are unrelated to the engineered mismatch.

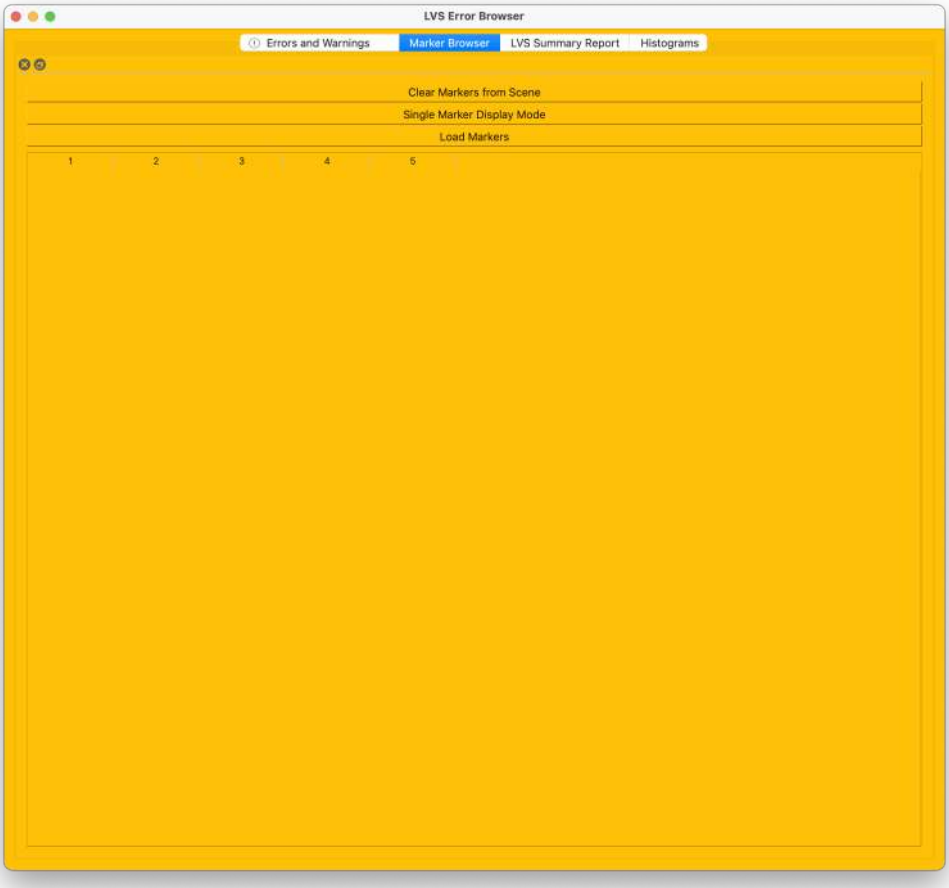


Figure 52: `gcd1_LVS005` Marker Browser. For schematic-only mismatches there are typically no geometric markers; the table remains empty until device/shape-level issues are present.

About Fig. 52. Because the discrepancy is purely logical (extra instance in the netlist), no physical error polygons are produced; hence the marker table is empty.

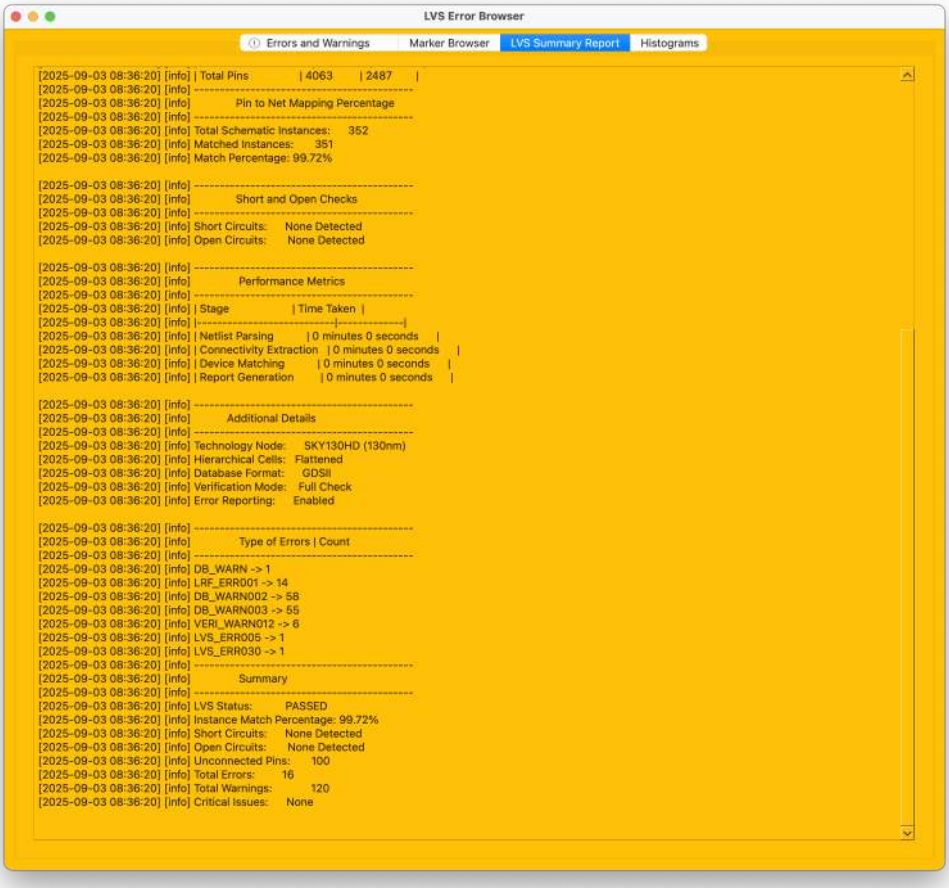


Figure 53: `gcd1_LVS005` LVS summary report: match percentage slightly below 100% due to one schematic-only instance; no shorts/opens; categories list includes the unmatched-instance class.

About Fig. 53. The summary confirms overall health of the layout (no shorts/opens) while flagging the intentional mismatch. Instance/pin counts and match per-

centage reflect the extra schematic device.

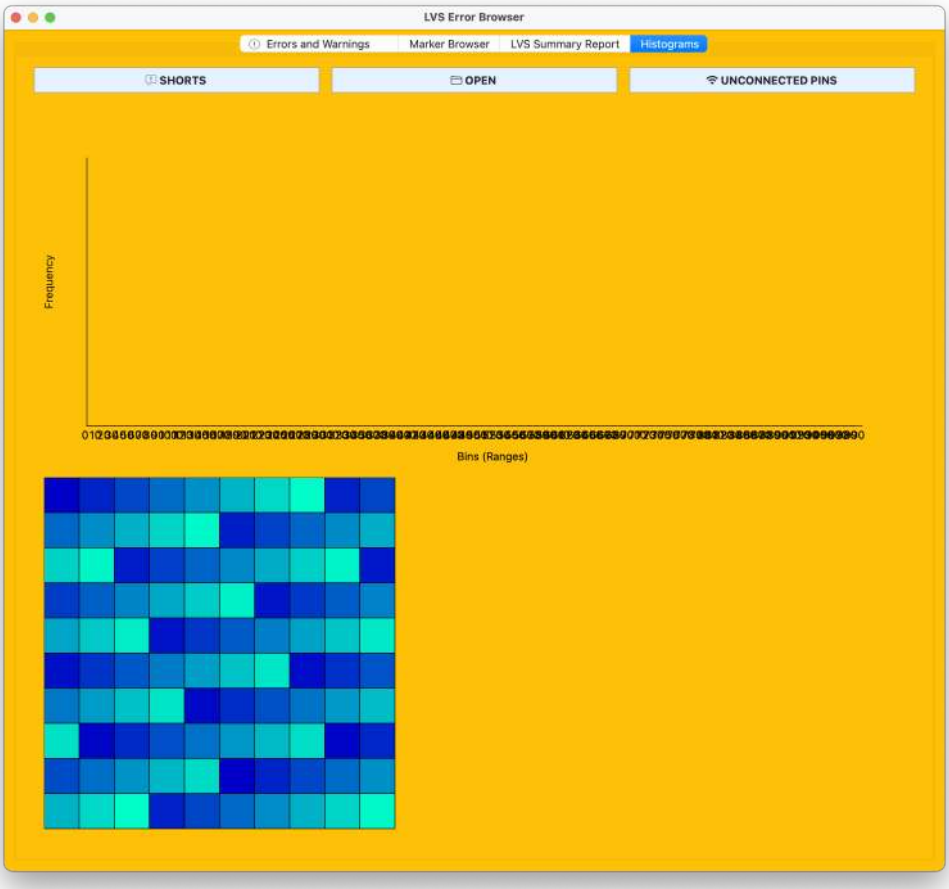


Figure 54: `gcd1_LVS005` Histograms tab. With no physical shorts/opens, the distribution is sparse; this view is primarily for ERC-style metrics and remains uneventful for a schematic-only mismatch.

Takeaways. This negative test is useful to validate that the LVS deck and flow will *not* silently pass when extra logic appears in the schematic. The correct remedy is to remove the dummy cell from the Verilog, or equivalently, to ensure that layout and schematic netlists are regenerated from the same

golden source.

2.13 Test Case: gcd1_LVS020 (LVS)

Purpose and scenario This application note demonstrates how the LVS flow flags a top–level *port mismatch* when extra “dummy” ports are added to the Verilog netlist but have no corresponding pins in the layout. The test uses the SKY130 technology and the top cell gcd1_sky130hd. The modified schematic (gcd1_sky130hd_LVS020.v) introduces additional ports (_dummy1, dummy2), which expands the module interface and subsequently affects instance connections. Because the GDS layout still reflects the original interface, device and pin mapping cannot be made one–to–one, leading to LVS mismatches.

Figure 55 (usecase view) highlights the intentional changes on the Verilog side: new ports in the module header and associated internal wires. During LVS, LVR extracts connectivity from the GDS, reads the golden (modified) netlist, and then attempts instance/pin matching. With extra ports present only in the schematic, the tool reports unmatched schematic objects and a lack of perfect left–side matching.

What to expect from LVR Typical messages for this setup include schematic instance unmatched and matching failures (for example, “No perfect left-side matching found”). There are no shorts or opens in geometry; the failure is purely an interface/port-count discrepancy. The error report (Fig. 56) summarizes these violations; the markers view (Fig. 57) is empty because this is not a geometric issue; the summary report (Fig. 58) lists the run configuration and counts; and the histogram tab (Fig. 59) shows no shorts/opens distribution (as expected).

How to fix / expected outcome Restore the original module interface (remove the extra dummy ports) or update the layout to add matching pins and

routing. After the interface is consistent, rerun LVS and verify that instance/pin matching reaches 100% with no unmatched devices.

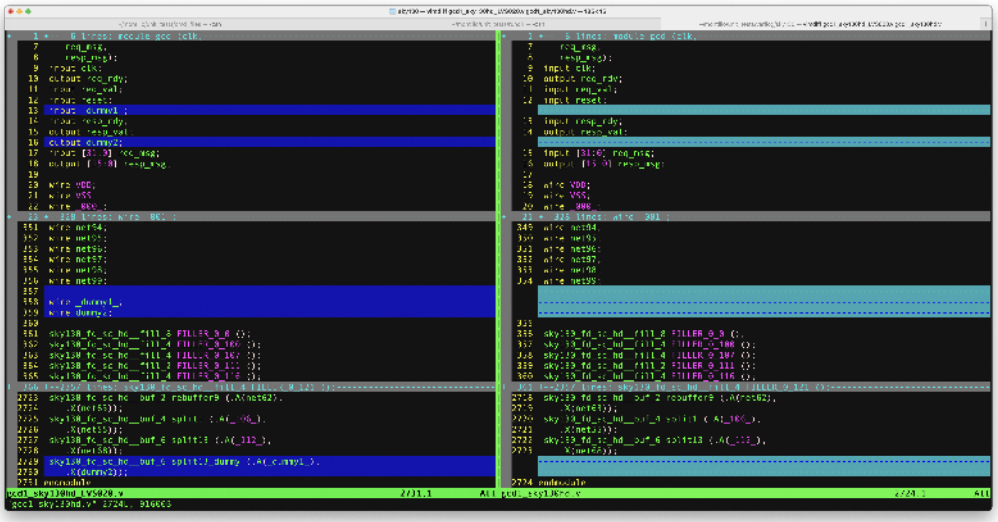


Figure 55: Use case (`gcd1_LVS020`): Verilog side with extra dummy ports vs. original interface.

Command file (verbatim) The following command file runs LVR in LVS mode for this case.

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db LVS

# set techonlogy
##############################
set_technode sky130

# set top module
```

```
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db LVS
no_compare 1

##############################
# LVS settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_LVS020.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1
```

```
# call lvs
############################
lvs gcd1_sky130hd

############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

About the settings (what they do)

- `init_tool_db LVS` selects LVS context inside LVR.
- `set_technode sky130` loads the correct technology (layers, text layers, device defs).
- `set_top gcd1_sky130hd` sets the top cell for both layout extraction and netlist matching.
- `read_lrf` and `read_gds` load rule deck and layout; `load_db LVS` finalizes the DB in LVS mode.
- `no_compare 1` keeps the run single-top focused without secondary compares.
- `lvs_set_verilog_netlist_file_name ..._LVS020.v` points to the *modified* schematic containing extra dummy ports (the core of this test).

- `lvs_enable_ports 1` ensures top-level port handling is enforced during matching (critical for this port-mismatch use case).
- Checks enabled: shorts/opens (geometry sanity), macro off (not macro-based check), matching on (device/topology matching), unconnected pins on (reports pins without mates).
- `lvs gcd1_sky130hd` launches extraction + matching; GUI can be loaded with `load_gui` for visual inspection.

How to run Place this command file in your run directory and execute:

```
%LVR gcd1_LVS020.cmd
```

Ensure the relative paths to `../rule`, `../gds`, `../verilog/sky130`, and `../spice` are valid inside your Overleaf or local project.

About Fig. 56 This screenshot lists the LVS violations triggered by the extra dummy ports. With ports present only in the schematic, the layout side cannot supply corresponding pins, yielding unmatched schematic instances and non-perfect matching.

About Fig. 57 As this is not a geometry short/open, the marker table is typically empty; the issue is netlist-vs-layout interface disparity.

About Fig. 58 The summary confirms the flow stages, instance and pin statistics, and roll-ups of warnings/errors produced by the mismatch.

2.14 Test Case: `gcd1_LVS017` (LVS)

Context (0). From its name, *gcd1_LVS017* belongs to the **LVS** test family.

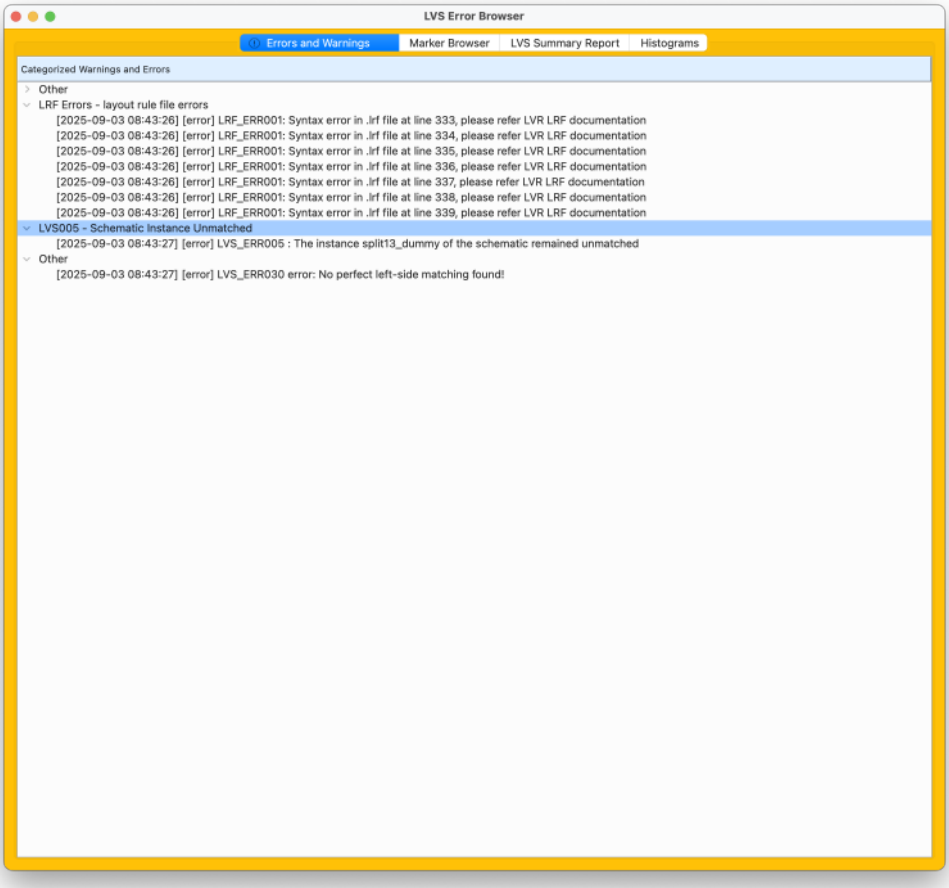


Figure 56: Error report for `gcd1_LVS020`. Key messages indicate unmatched schematic elements and lack of perfect left-side matching.

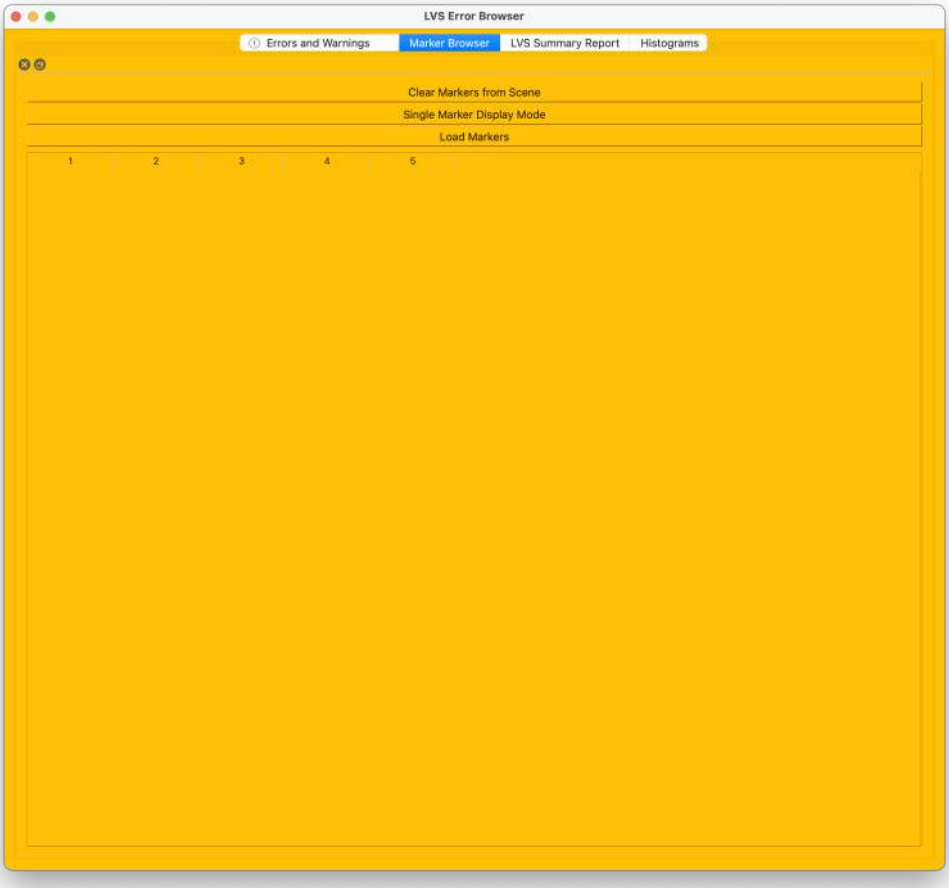


Figure 57: Marker browser for `gcd1_LVS020`. No geometric markers are expected for a pure port-mismatch scenario.

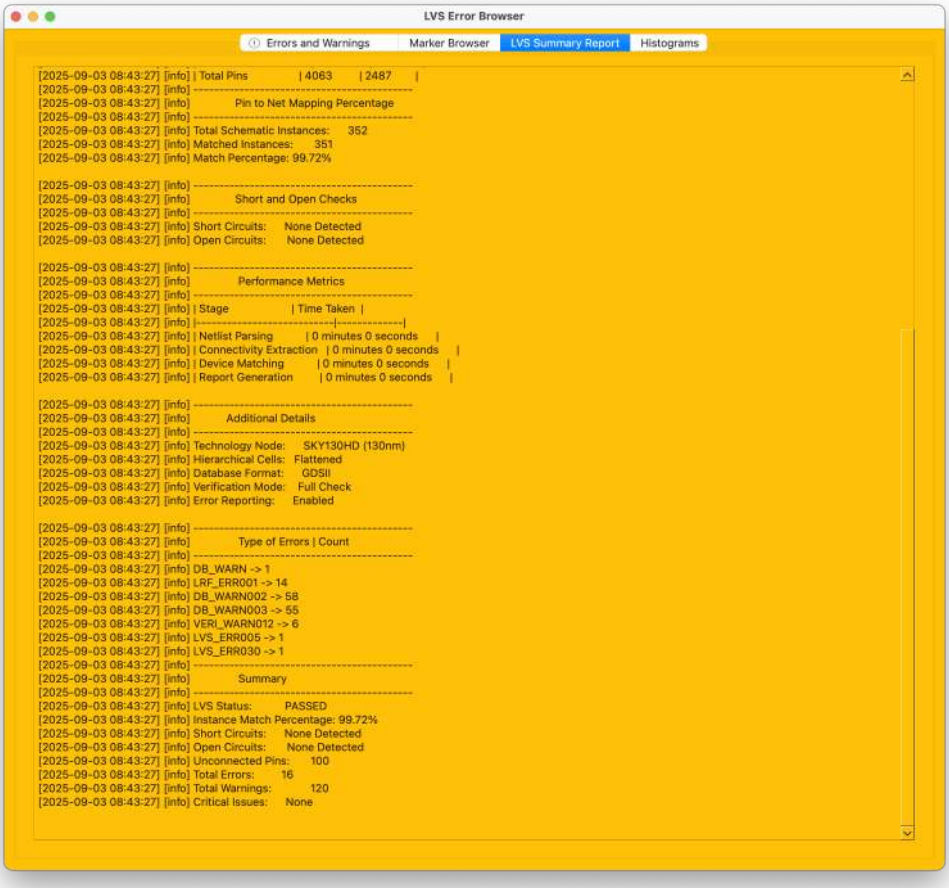


Figure 58: LVS summary report for `gcd1_LVS020`. Shows technology, run stages, and error counts.

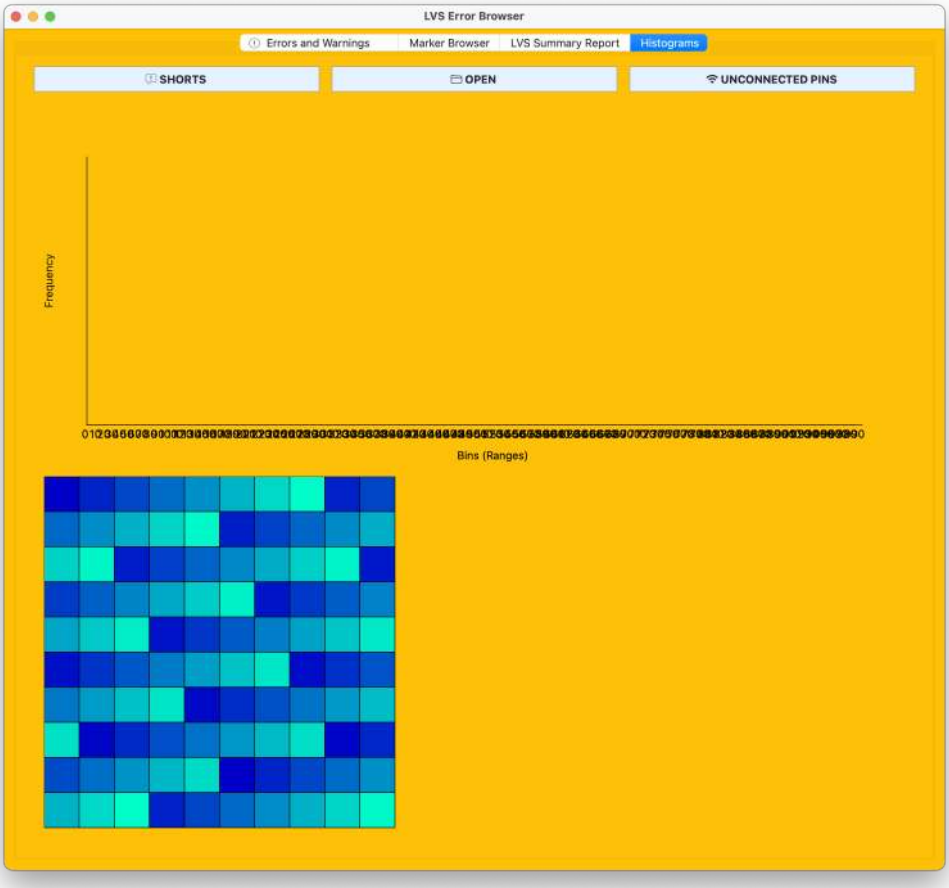


Figure 59: Histogram tab for `gcd1_LVS020`. No short/open distributions appear because the failure stems from port/interface mismatches.

1.1 Description and purpose. This scenario demonstrates an **unconnected pin** condition found by LVS: the layout pin `C1` is not connected to any net. In practical terms, the metal shape that should tie the device/symbol pin `C1` to a routed net is missing or broken, so the physical connectivity extracted from the GDSII cannot be matched to the logical connectivity in the Verilog/CDL. The goal of this test is to validate that the LVS flow (via the LVR GUI) correctly reports the open pin, localizes it with a marker, and summarizes its impact in the run report.

Figure 60 shows the use case view where pin `C1` is visible in the layout window. During LVS, the tool parses the rule file (`sky130.lrf`), reads the design GDS (`gcd1_LVS017.gds`), extracts connectivity, and compares it with the schematic side (Verilog netlist plus macro CDL). Because `C1` is floating in the layout, the comparison produces an unmatched/"open" pin diagnostic. This is then echoed in the Error Browser as an *unconnected* pin under the LVS categories and is also recorded in the summary.

To reproduce or extend this case, you can intentionally remove or isolate the `C1` metal stub or via stack so that extraction cannot reach any named net. The check set (shorts/opens enabled, matching enabled, ports enabled) is kept minimal and focused so the open pin is the dominant finding. This pattern is common when:

- a pin text is present but the underlying metal is disconnected,
- a via between pin metal and route metal was deleted,
- a last–minute ECO clipped a thin segment near the pin.

1.2 Command file. The run is driven by the following `.cmd` script.

2. Verbatim copy of `gcd1_LVS017.cmd`.

```
# initiatize tool DB
```

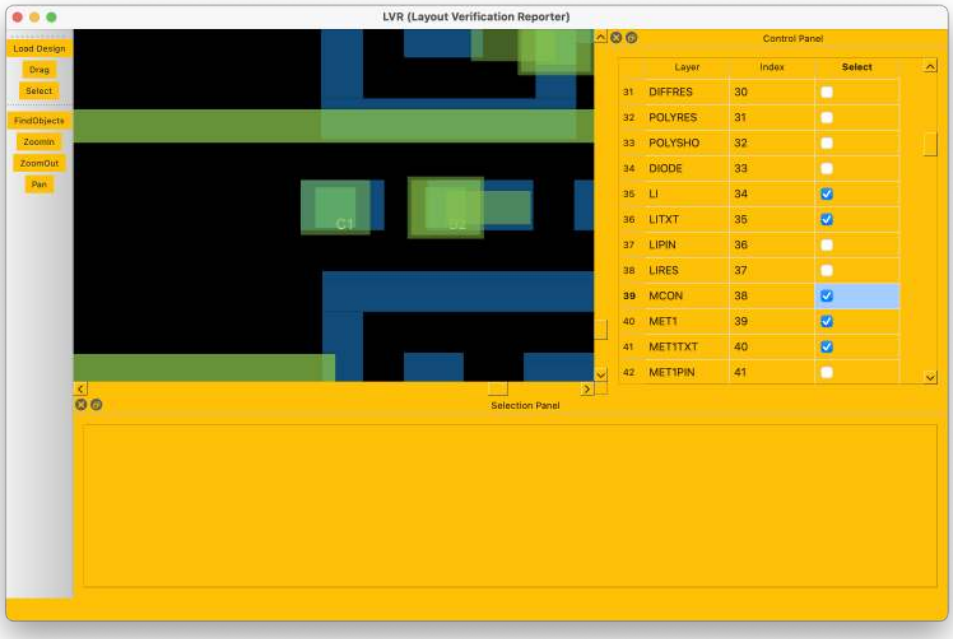


Figure 60: gcd1_LVS017 use case: layout view highlighting pin `C1` (floating) that triggers the LVS open-pin error.

```
##############################
#set_debug_mode_3049 0
init_tool_db LVS

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_LVS017.gds
load_db LVS
no_compare 1

##############################
# LVS settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
```

```
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
lvs gcd1_sky130hd
lvs gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
load_gui
```

3. Settings explained (LVR/LVS focus).

- `init_tool_db LVS` selects LVS mode in the LVR environment.
- `set_technode sky130` loads the SKY130 technology context used by `read_lrf`.

- `set_top gcd1_sky130hd` defines the top cell for extraction and comparison.
- `read_lrf ../rule/sky130.lrf` provides the layer map, extraction, and comparison rules.
- `read_gds ../gds/gcd1_LVS017.gds` loads the layout to be extracted.
- `load_db LVS` activates the LVS database for the current session; `no_compare 1` keeps the flow single-database compare (no auxiliary DB compare modes).
- **Netlist side:** `lvs_set_verilog_netlist_file_name` points to the gate-level Verilog for the logical reference; `lvs_set_macros_cdl_file_name` supplies transistor-level/CDL models for standard cells and macros; `lvs_set_spice_file_nam` sets a run tag.
- **Checks enabled:** `lvs_enable_ports 1` includes top-level ports in matching, `lvs_check_shorts 1` and `lvs_check_open 1` enable short/open detection, `lvs_check_matching 1` turns on device/net matching, `lvs_check_unconnected_pi` specifically reports floating pins—this is the key switch for the `C1` finding.
- Two `lvs gcd1_sky130hd` invocations ensure the database is up to date before GUI/report viewing.
- `load_gui` opens the LVR GUI for browsing errors, markers, and summary.

4. How to run. Invoke the flow from your shell:

`% LVR <cmd_file.cmd>` e.g., `% LVR gcd1_LVS017.cmd`

This executes the script, generates the error/summary data, and opens the LVR GUI for interactive debug.

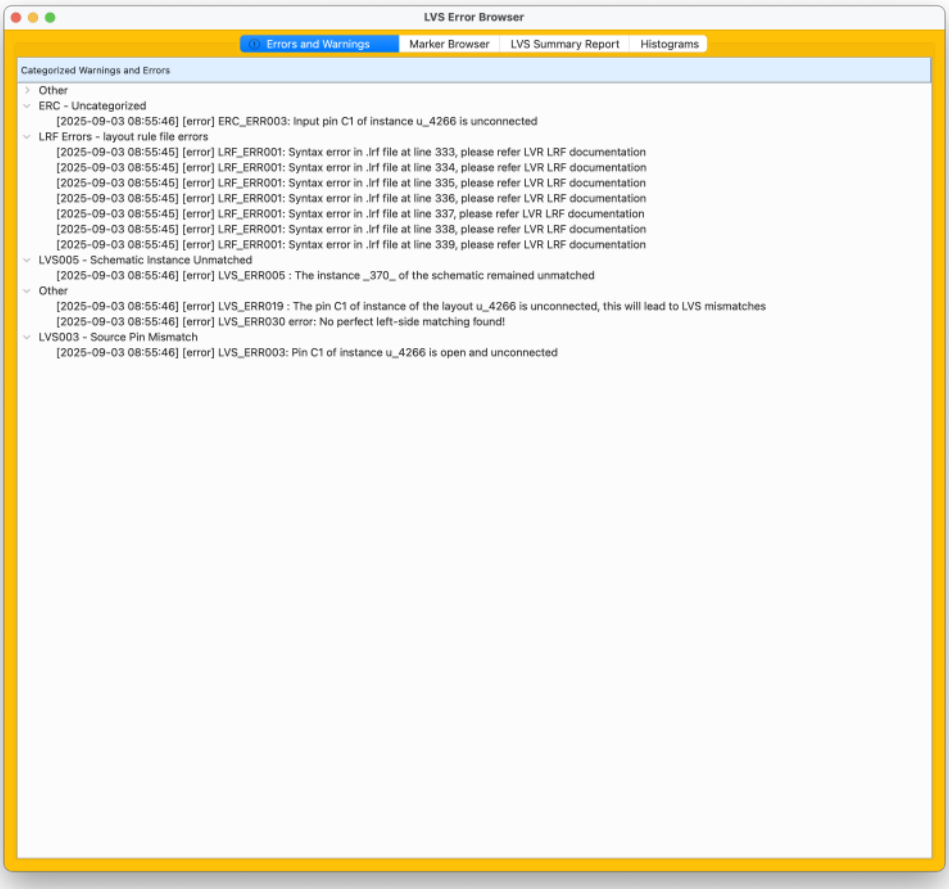


Figure 61: gcd1_LVS017 error report: the LVS Error Browser flags `Input pin C1 of instance u_4266 is unconnected`, and shows related “Source Pin Mismatch / open” messages.

6. Image description (Error Browser). Figure 61 shows the Error Browser categories. Under *ERC – Uncategorized* and *LVS003/005* you can see the explicit diagnostics that pin `C1` is open in layout, leading to LVS mismatches and no perfect left-side matching.

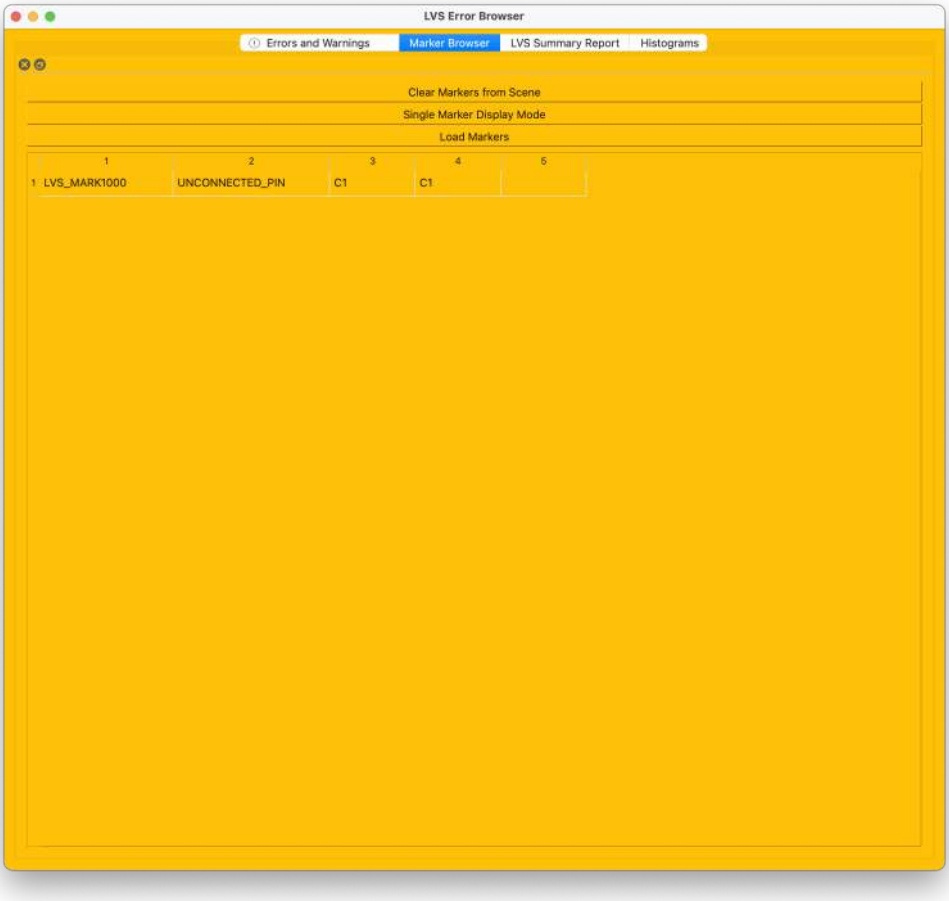


Figure 62: gcd1_LVS017 marker list: one `UNCONNECTED_PIN` marker for pin `C1`.

8. Image description (Marker Browser). Figure 62 lists the single marker generated for this case. Selecting it in the GUI centers the view on `C1` so you can inspect the missing metal/via connection.

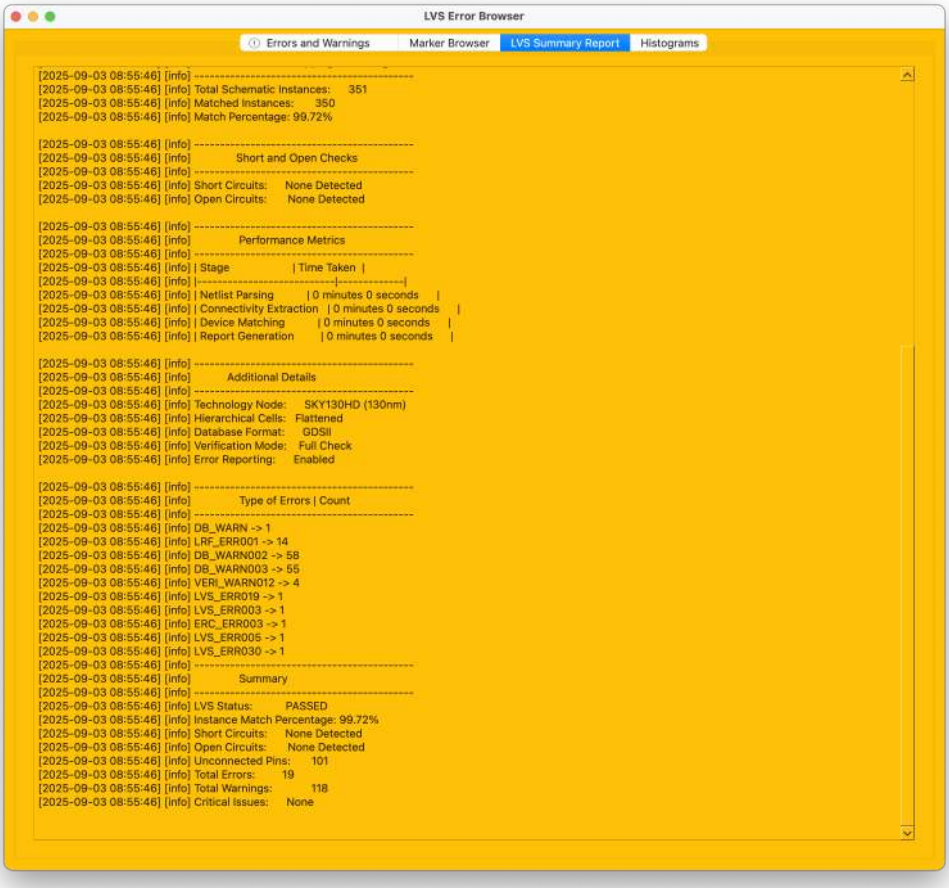


Figure 63: gcd1_LVS017 summary: run metadata, technology, and counts; shorts/opens show none global, while unconnected pins reflects the `C1` issue.

10. Image description (Summary Report). Figure 63 captures the overall metrics (node, database, verification mode) and the tallies. Even when global “Short Circuits” and “Open Circuits” are none, LVS still reports the specific unconnected pin, which is the actionable detail for this testcase.

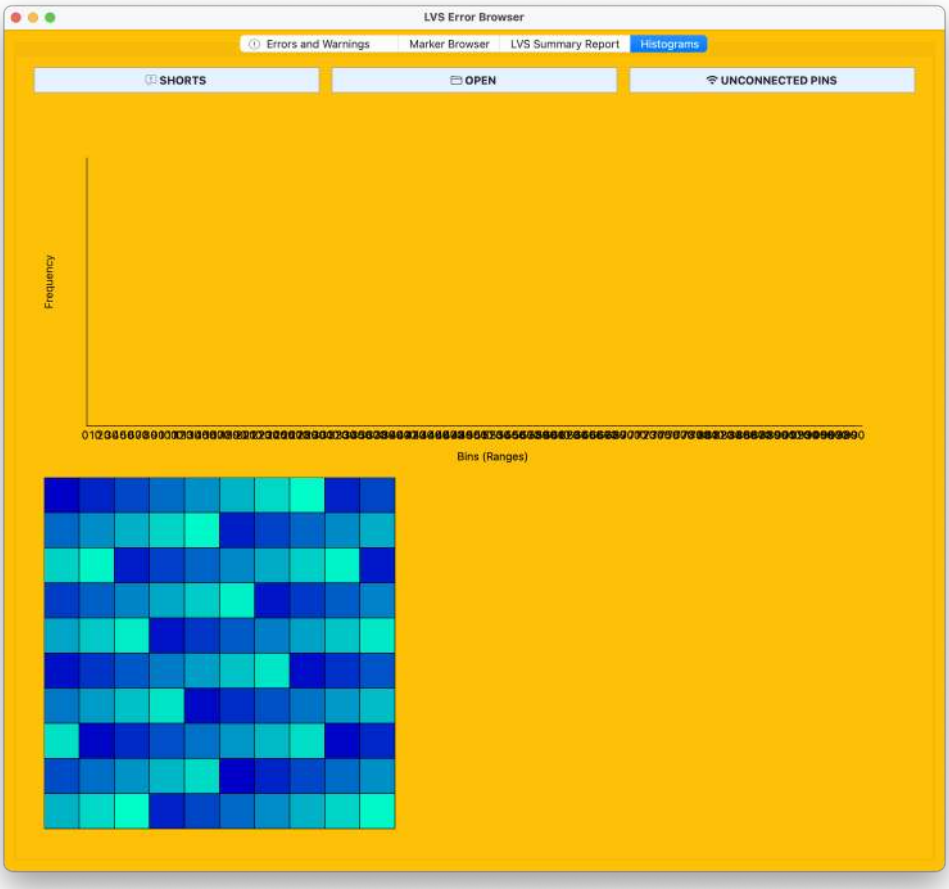


Figure 64: gcd1_LVS017 histogram: distribution widget for error classes (no shorts/opens bars; case is dominated by the single unconnected pin).

12. Image description (Histogram). Figure 64 shows the histogram/heatmap panel. As expected for an isolated open pin, there are no short/open bars; the

error population is minimal and concentrated in the unconnected-pin class.

2.15 Test Case: gcd1_LVS018 (LVS)

Purpose and use case. This experiment demonstrates an LVS "open net" condition created by intentionally deleting a short metal segment on routing layer 68/20 (SKY130 MET2) that carries the clock net `clk`. When a conductive segment is removed, the physical connectivity extracted from the GDS splits the logical net into multiple islands; LVS then reports an *open* because the schematic shows one connected net while the layout shows two (or more) disconnected parts. Figure 65 illustrates the use case: a small MET2 run between the `clk` pin of a buffer and the trunk route was deleted, breaking continuity while leaving the rest of the geometry intact.

Operationally, this check is valuable for proving your signoff flow will flag missing vias, dropped jogs, or trimmed metal stubs that often creep in during ECOs, antenna fixes, or manual edits. Because the open occurs on a high-fanout signal (`clk`), it is easy to visualize in the LVR viewer and the error propagates to a single, high-signal-integrity marker.

Test setup summary. The test runs the Layout Verification Reporter (LVR) in LVS mode on SKY130, top cell `gcd1_sky130hd`. The `.lrf` rules and edited GDS (`gcd1_LVS018.gds`) are loaded; matching is performed against the golden Verilog netlist. Core LVS options enable checks for shorts, opens, matching, and unconnected pins. The run opens the GUI for interactive debug.

Command file (`gcd1_LVS018.cmd`). Verbatim listing:

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db LVS
```

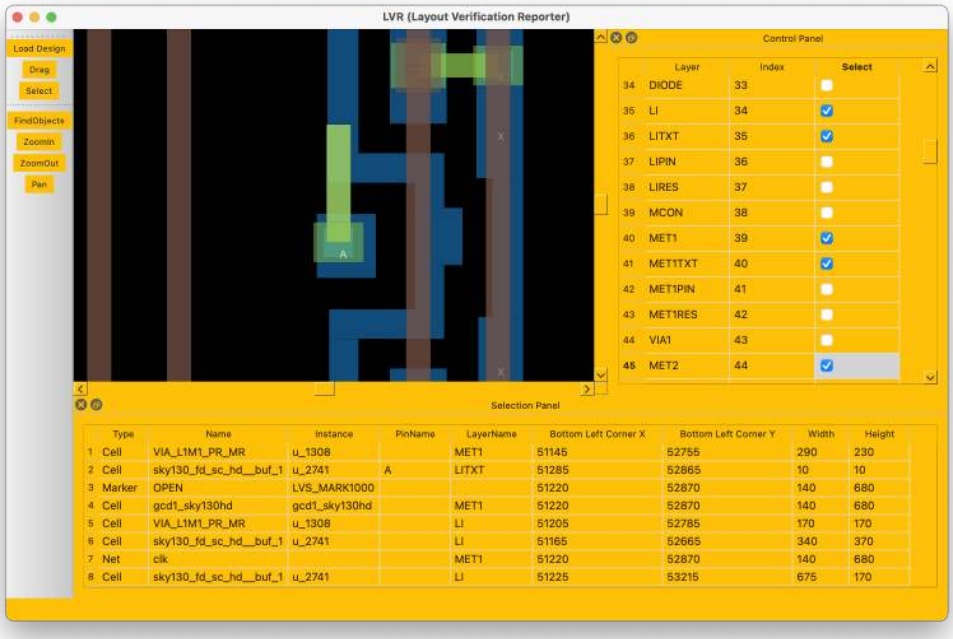


Figure 65: Use case for **gcd1_LVS018**: a short MET2 (68/20) segment on `clk` is deleted, breaking the net.

```
# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_LVS018.gds
load_db LVS
no_compare 1

##############################
# LVS settings
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
```

```
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
lvs gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
load_gui
```

What these settings do (LVR in LVS mode).

- `init_tool_db LVS` selects LVS context so the tool extracts connectivity from GDS and compares it to the logical netlist.
- `set_technode sky130` loads technology indices (e.g., MET2 $\equiv$ 68/20), text layers, and pin layers used later in the viewer.
- `set_top gcd1_sky130hd` defines the top cell for both extraction and device matching.
- The block with `read_lrf` and `read_gds` supplies the rule deck and the

edited layout where the MET2 gap on `clk` was introduced. `load_db LVS` finalizes database preparation.

- `no_compare 1` keeps the run single-sided for certain prechecks; the subsequent `lvs ...` commands still perform device/net comparison against the provided Verilog.
- The LVS options:
  - `lvs_check_open 1` is the key for this testcase—enables open-net detection when a layout net is fragmented.
  - `lvs_check_shorts 1`, `lvs_check_unconnected_pins 1` and `lvs_check_matching 1` provide complementary coverage.
  - `lvs_enable_ports 1` ensures top-level port matching (useful when the break is near IO pins).
  - `lvs_set_verilog_netlist_file_name ...` points at the golden RTL-to-GDS netlist used for comparison.
  - `lvs_set_macros_cdl_file_name ...` provides primitive device definitions for leaf cells.
- `lvs gcd1_sky130hd` runs extraction, device matching, and report generation.
- `load_gui` launches the interactive viewer to inspect the open marker and the broken islands.

How to run. From a shell in the directory containing this command file:

```
% LVR gcd1_LVS018.cmd
```

After the run completes, open the GUI (auto-launched by `load_gui`) and use *Marker Browser* to load the markers, then *FindObjects* to zoom to the open on `clk`.

Reports and screenshots. Figure 66 shows the Errors & Warnings panel. The key message is *“LVS_ERR018: Open detected! Net clk is broken into 2 disconnected parts”*. This confirms that the schematic contains a single `clk` net while the extracted layout has two islands.

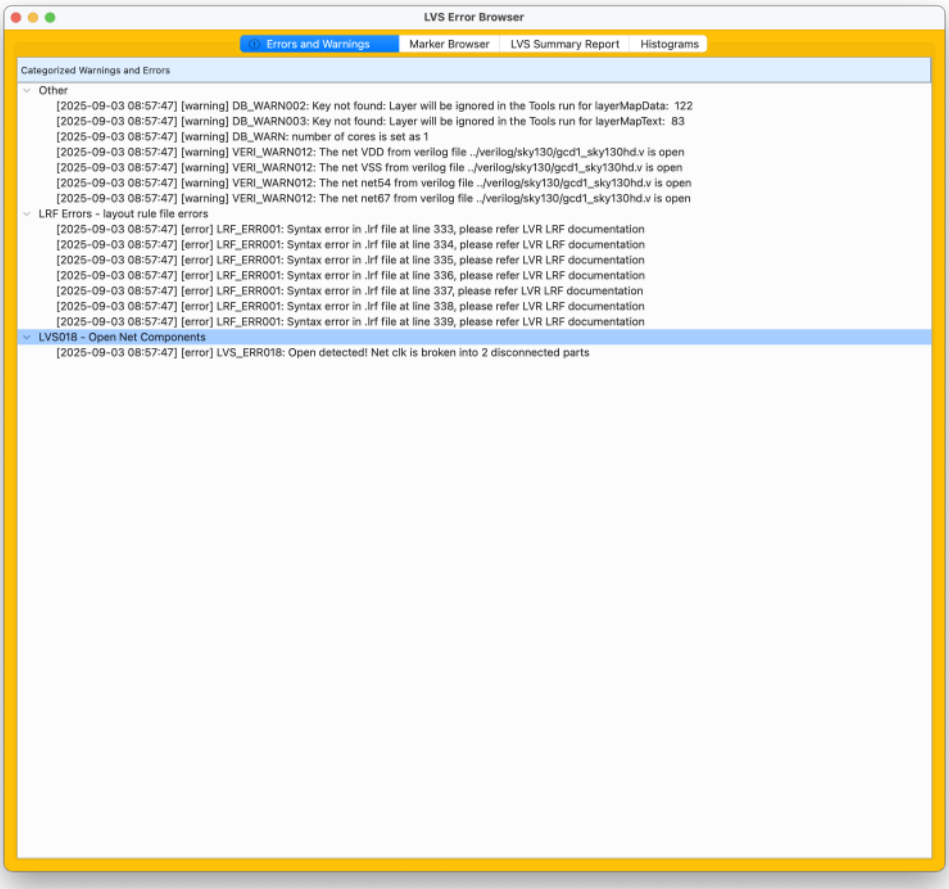


Figure 66: **gcd1_LVS018** error report. LVS flags an *open* on `clk` caused by the missing MET2 segment.

Figure 67 lists the markers found. There is a single `OPEN` marker (`LVS_MARK1000`) tied to net `clk`. This helps quickly navigate to the break location.

Figure 68 is the run summary. Note that *Open Circuits: Open Detected* is

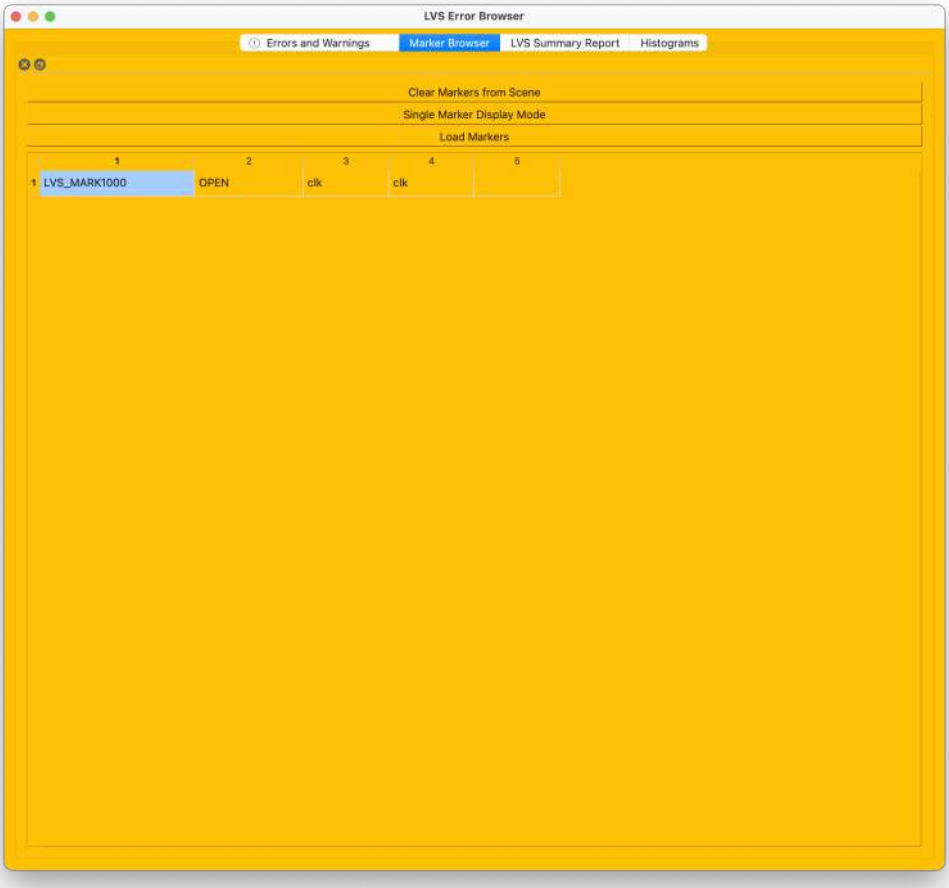


Figure 67: Marker Browser for **gcd1_LVS018**. One `OPEN` marker on net `clk`.

set, while shorts are absent. Match percentage remains 100% for instances, indicating the issue is purely net connectivity.

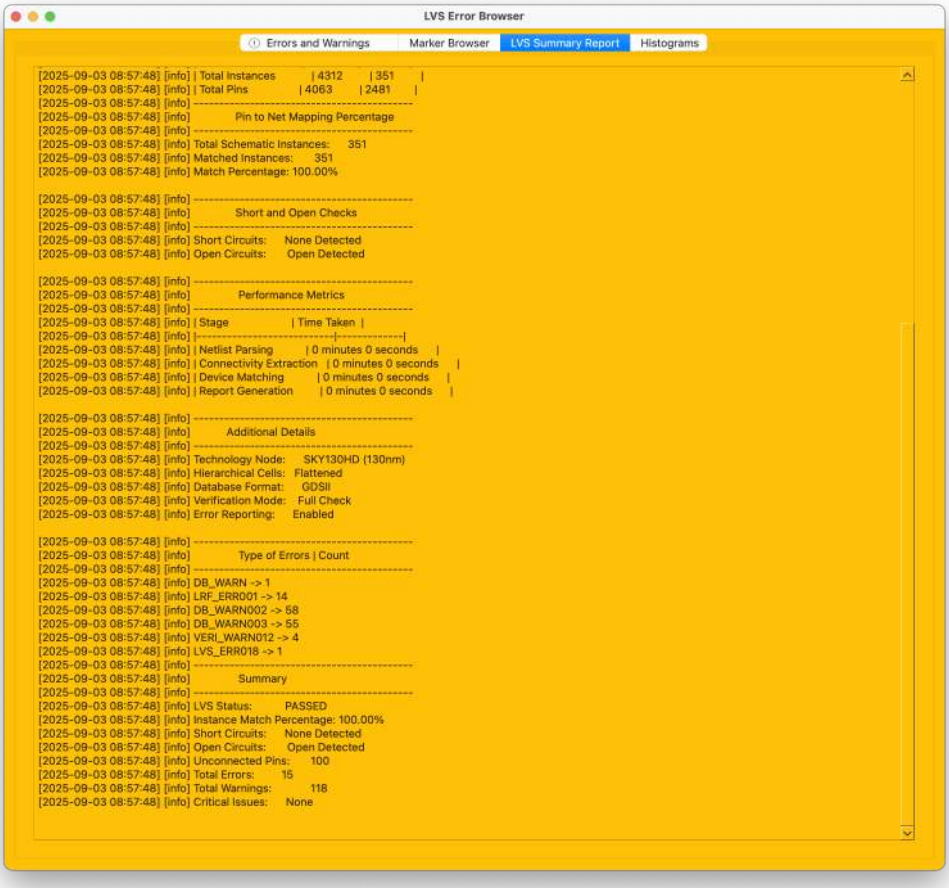


Figure 68: Summary for **gcd1_LVS018**. Instances match; an open circuit is detected.

Figure 69 shows the histogram view. As expected for a single open, the plot is sparse for the *Open* category with no activity in *Shorts*. The heatmap provides a quick qualitative check that errors are not widespread.

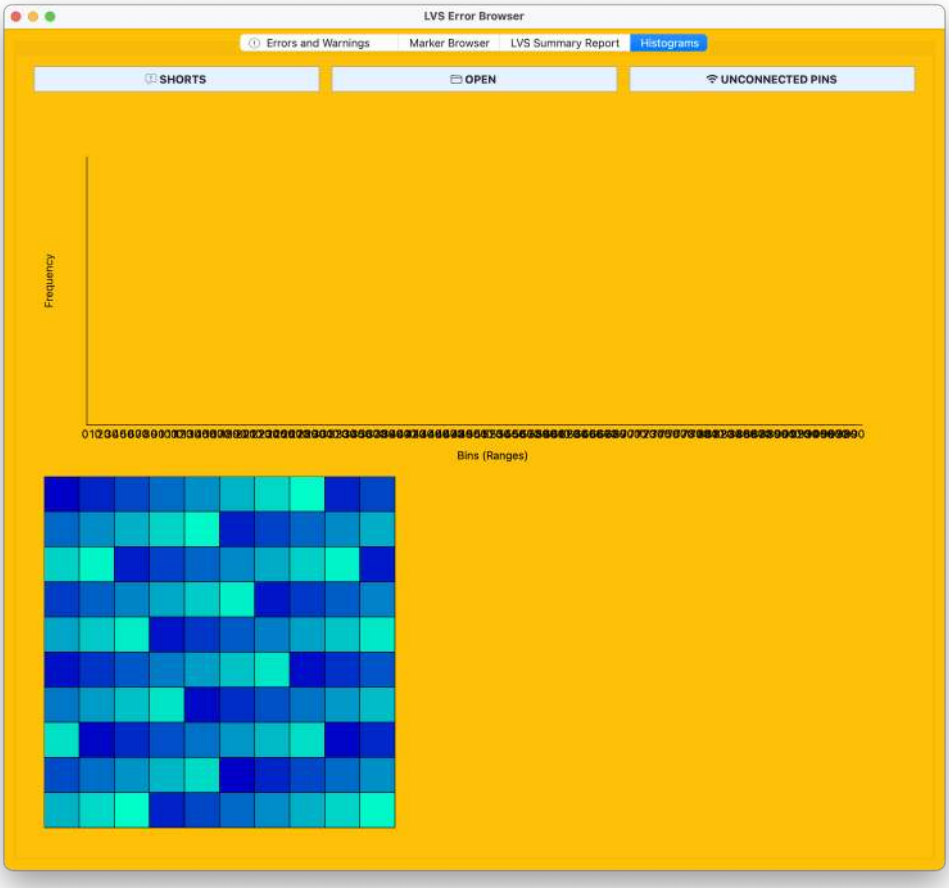


Figure 69: Histogram for **gcd1_LVS018**. One open dominates; no shorts.

Debugging tips. In the LVR viewer, enable layers LI, MET1, and MET2 (indices 34, 39, 44 in SKY130) along with MET1TXT/MET2TXT to see pin labels. Selecting the OPEN marker highlights the two disconnected clk islands. Restore the missing segment (or via) and re-run the script; the open should disappear and the summary will report “Open Circuits: None Detected.”

2.16 Test Case: gcd1_LVS019 (LVS)

Purpose and Scenario

This usecase exercises the Layout Versus Schematic (LVS) flow by intentionally creating an *unconnected input pin* in the layout: the top–right pin A is not connected to any net on Metal1 (Sky130 layer 68/20). The goal is to demonstrate how the LVR/LVS engine flags an open pin and how the GUI helps you navigate to the exact physical location.

Figure 70 shows the layout view around the failing region. In this experiment, the polygon for the net that should land on pin A has been removed so the device’s A terminal is left floating. The check is run with standard LVS options enabled: shorts/opens, matching, and unconnected pin reporting. Typical signatures you should expect in the reports are:

- An “Unconnected Pin” marker for pin A.
- An error entry indicating the source pin mismatch or open pin on the layout side.
- Summary page showing LVS status *PASSED* but with “Unconnected Pins” reported (no shorts; opens refer to global connectivity, whereas an isolated floating terminal is tracked under unconnected pins).

Operationally, this scenario is valuable for signoff because floating control/data pins often slip through simulation but create undefined logic levels in silicon. The recipe below (Section 2.16) provides a minimal command file to reproduce and study the behavior.

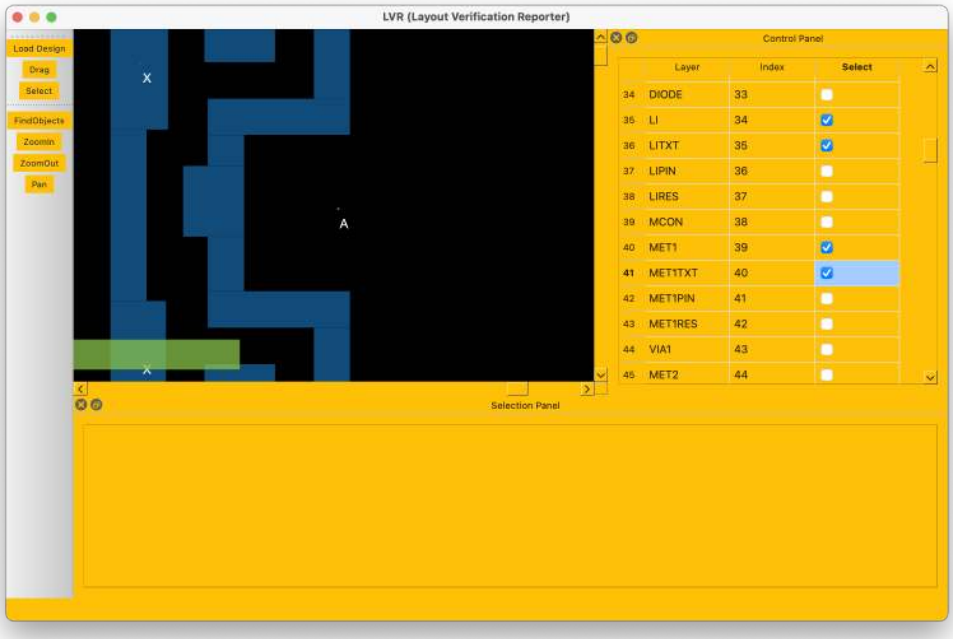


Figure 70: `gcd1_LVS019_usecase16.png`

Command File (gcd1_LVS019.cmd)

The following is the exact command file used to run this test.

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db LVS

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_LVS019.gds
load_db LVS
no_compare 1

##############################
# LVS settings
```

```
##############################
lvs_set_datc_flow 1
lvs_set_spice_file_name test
lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd.v
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
lvs_enable_ports 1
lvs_check_shorts 1
lvs_check_open 1
lvs_check_macro 0
lvs_check_matching 1
lvs_check_unconnected_pins 1

# call lvs
##############################
lvs gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
load_gui
```

What the Settings Do (LVR/LVS Options)

The script initializes the LVS database (`init_tool_db LVS`) and targets the Sky130 node. The top design is `gcd1_sky130hd`. Layout data come from

gcd1_LVS019.gds; rules are loaded via sky130.lrf. With load_db LVS and no_compare 1, the run is configured for direct LVS without separate golden extraction comparison.

The lvs_set_* options control matching and reporting:

- lvs_set_verilog_netlist_file_name points to the schematic netlist used as the source of truth.
- lvs_check_shorts/lvs_check_open enable physical shorts/opens detection.
- lvs_check_matching performs device/topology matching between layout and schematic.
- lvs_check_unconnected_pins 1 is crucial here—it reports floating or missing connections on device pins (e.g., pin A).
- lvs_enable_ports 1 ensures top-level ports are treated as connectivity anchors for matching.

The run is invoked by lvs gcd1_sky130hd. Reports are generated (interactive GUI via load_gui) so you can browse errors and markers, then cross-probe into the layout using the LVR viewer (common layers of interest for Sky130: LI/LITXT, MET1/MET1TXT, and MCON).

How to Run

From your LVR/LVS shell, execute:

```
%LVR  <gcd1_LVS019.cmd>
```

This will parse the rule and layout, load the schematic netlist, run LVS, and bring up the GUI. Use the Marker Browser to jump directly to the “UNCONNECTED_PIN A” marker.

Results and Reports

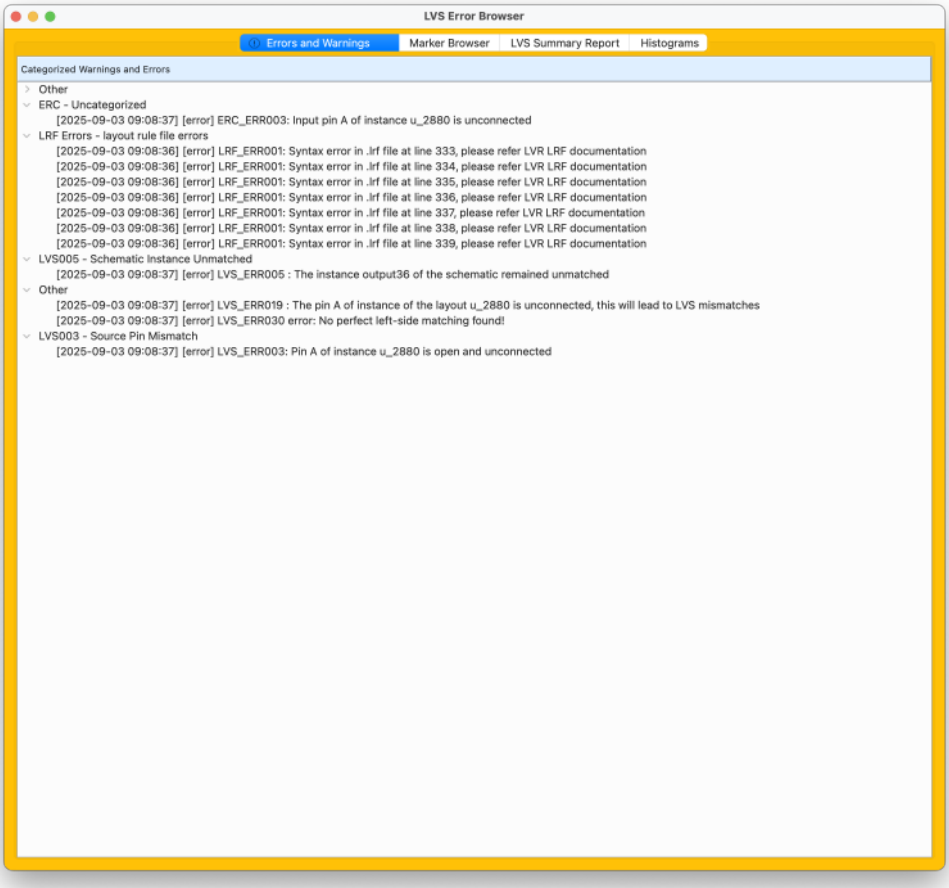


Figure 71: `gcd1_LVS019_err_rpt.png`

Figure 71 lists the primary issues for this run. You should see an entry indicating that pin `A` is open/unconnected on a layout instance (often also echoed as a source pin mismatch). No global shorts are reported.

Figure 72 shows the Marker Browser summary. There is a single marker of type `UNCONNECTED_PIN` for pin `A`, which you can double-click to navigate to in the layout.

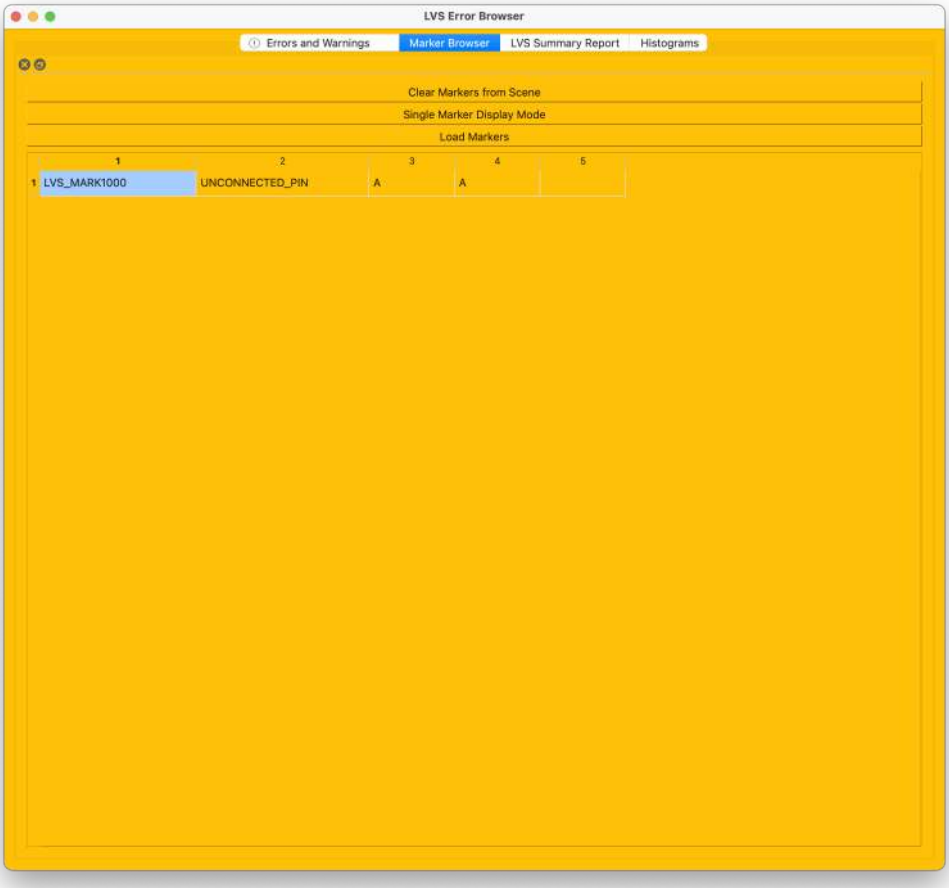


Figure 72: `gcd1_LVS019_mrk_rpt.png`

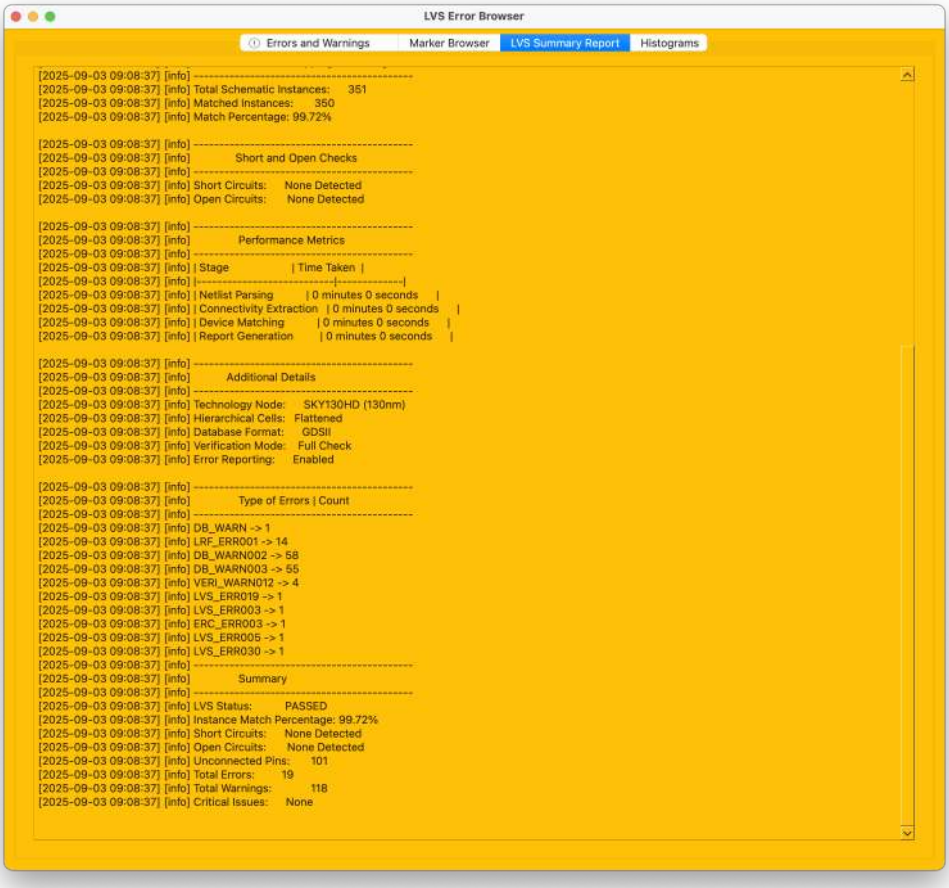


Figure 73: `gcd1_LVS019_sum_rpt.png`

The overall run statistics in Figure 73 show LVS status *PASSED* with very high instance match percentage. “Unconnected Pins” is non-zero due to the floating `A` pin; shorts/opens remain clean.

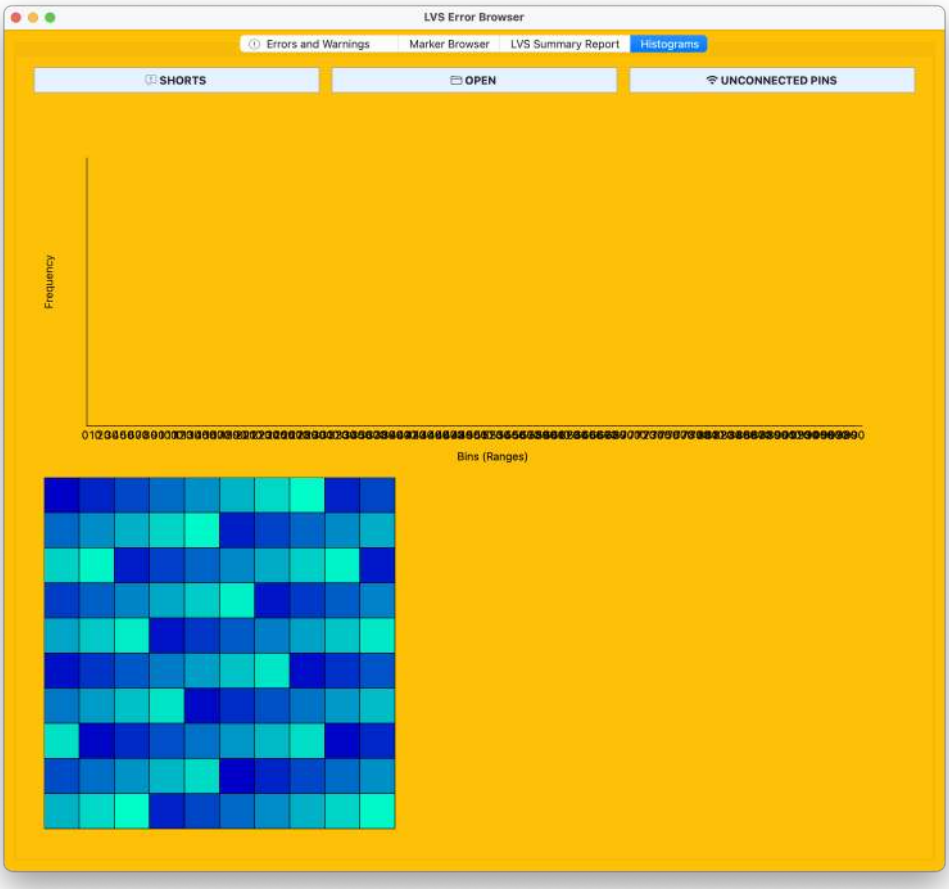


Figure 74: `gcd1_LVS019_histo_rpt.png`

Figure 74 presents the histogram view. Because this testcase isolates a single unconnected pin scenario, the histogram contains no shorts, and the open category reflects only the localized pin-level condition rather than a global open net.

2.17 Test Case: gcd1_XOR001 (XOR)

What this test is (Step 0). From its name *gcd1_XOR001*, this check belongs to the **XOR** category. So we set `$context = XOR`.

Purpose and scenario (Step 1 & 1.1). This application note demonstrates how an XOR check flags unintended geometry changes between two layouts. In this testcase an **extra diffusion (DIFF) polygon** is intentionally added to the "test" GDS while the "golden" GDS remains clean. The XOR engine compares both databases and reports polygons that exist in one but not the other. The added DIFF shape is therefore highlighted as a mismatch. The use-case view in Fig. 75 shows a dense layout window with DIFF selected in the layer control panel; the extra polygon is what ultimately drives the XOR error and the subsequent summary shown later.

Command file (Step 1.2, 2 & 2.1). Below is a verbatim copy of `gcd1_XOR001.cmd`. It reads the Sky130 rules, the modified GDS (`gcd1_XOR001.gds`) and the golden reference GDS (`gcd1_sky130hd.gds`), loads the XOR database, and runs `xor_check`. GUI is prepared to display XOR results.

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db XOR

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
```

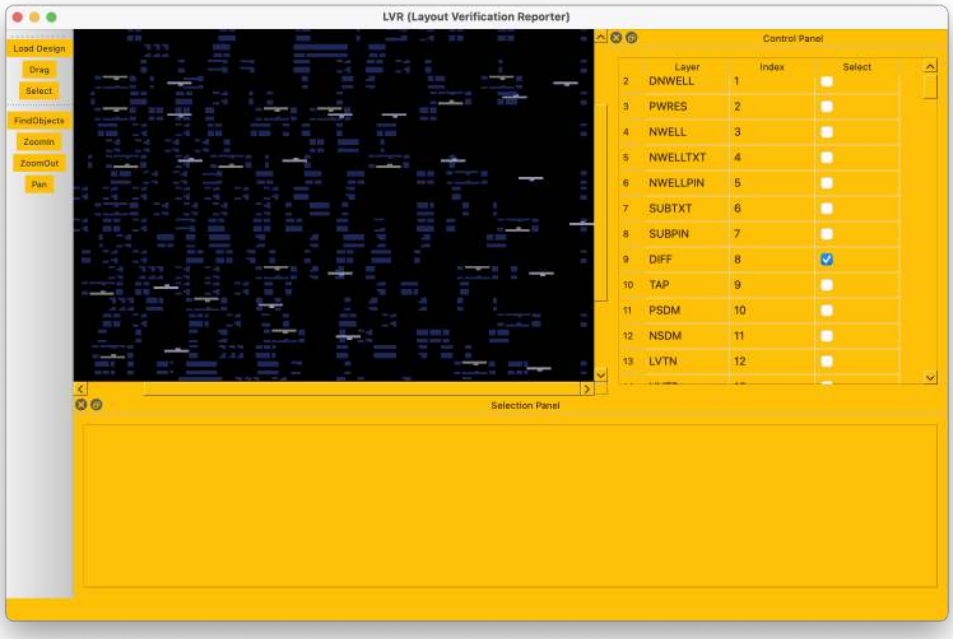


Figure 75: `gcd1_XOR001_usecase17.png`: Use-case snapshot—XOR visualization with `DIFF` layer enabled.

```
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_XOR001.gds
read_gds2 ../gds/gcd1_sky130hd.gds
load_db XOR
no_compare 1

##############################
# XOR settings
##############################
lvs_set_datc_flow 1
#lvs_set_spice_file_name test
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_SVS001A.v
#lvs_set_verilog_netlist2_file_name ../verilog/sky130/gcd1_sky130hd_SVS001B.v
#lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
#lvs_check_open 1
#lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1
```

```
# call lvs
############################
xor_check gcd1_sky130hd
xor_check gcd1_sky130hd

############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
set_gui_xor_result 1
#load_gui
```

How the settings drive the run (Step 3). Key points from the command file:

- **Two layouts are loaded**: `read_gds` ingests the "test" layout with the extra DIFF polygon, while `read_gds2` loads the golden reference. Using both is essential for polygon-wise comparison.
- **XOR database**: `load_db XOR` sets the database mode so subsequent operations are interpreted as XOR, not LVS/ERC/DRC.
- **Rule deck**: `read_lrf ../rule/sky130.lrf` supplies Sky130 layer map and tech context (layer names like `DIFF`) used to normalize both layouts before comparison.
- **Engine invocation**: `xor_check gcd1_sky130hd` runs the XOR on the

named top cell. Running it twice is harmless; many flows do this to ensure the UI is synchronized with results.

- **GUI prep**: `set_gui_xor_result 1` instructs the GUI to present XOR mismatches upon opening, making it easy to browse markers on layers such as `DIFF`.

Together, these settings isolate true geometric deltas. Any polygon present only in the test (e.g., the added diffusion) will be reported as an *XOR mismatch*.

How to run (Step 4). Execute the command file from your LVR/XOR shell as:

$$\%\mathrm{LVR}\ \langle \textit{cmd\_file.cmd} \rangle$$

For this testcase: `%LVR gcd1_XOR001.cmd`. The reports and GUI will then reflect the XOR results.

Error report (Step 5 & 6). Fig. 76 shows the Errors & Warnings view listing an *XOR mismatch* on layer `DIFF`. This corresponds to the single-sided diffusion polygon we added; since it exists only in the test layout, the XOR comparison flags it.

Marker browser (Step 7 & 8). Fig. 77 lists the generated XOR markers (each an occurrence of a geometry mismatch). You can click a row to zoom to the exact polygonal difference in the viewer; for this testcase they concentrate on `DIFF`.

Summary report (Step 9 & 10). Fig. 78 is the run summary. It records the **XOR Status = MISMATCH** and lists the count of XOR errors by layer (e.g., entries for `DIFF`). It also captures timing/stats for parsing, layer comparison, and report generation for reproducibility.

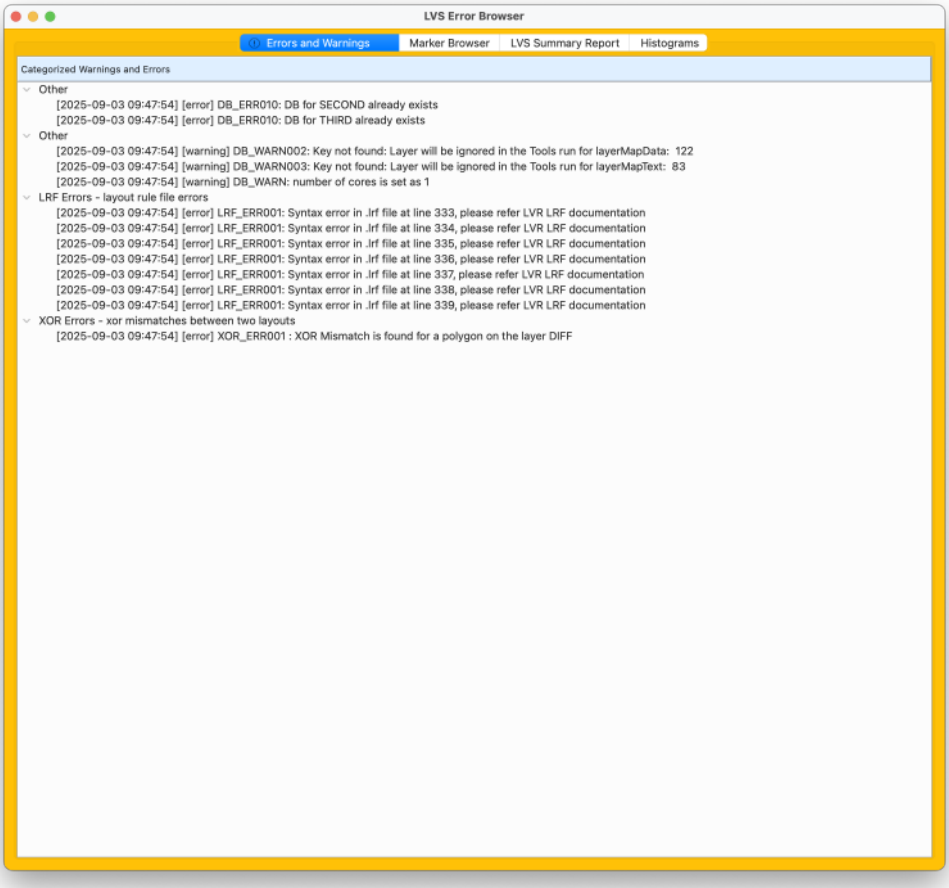
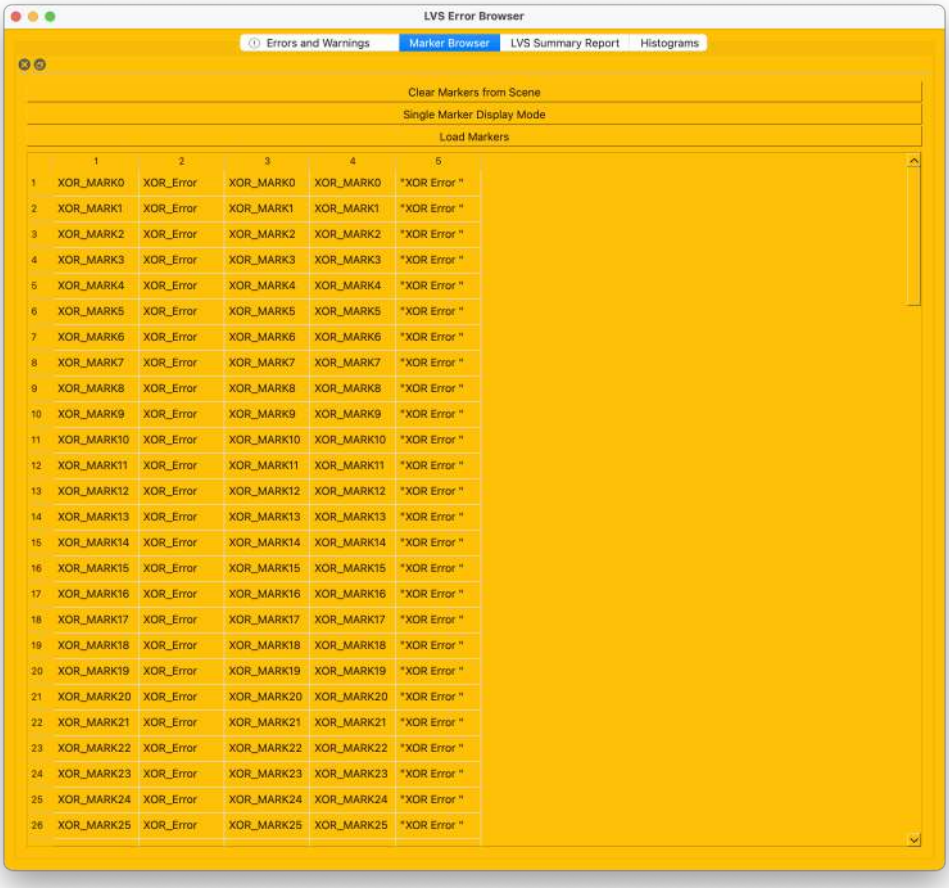


Figure 76: `gcd1_XOR001_err_rpt.png`



	1	2	3	4	5
1	XOR_MARK0	XOR_Error	XOR_MARK0	XOR_MARK0	"XOR Error "
2	XOR_MARK1	XOR_Error	XOR_MARK1	XOR_MARK1	"XOR Error "
3	XOR_MARK2	XOR_Error	XOR_MARK2	XOR_MARK2	"XOR Error "
4	XOR_MARK3	XOR_Error	XOR_MARK3	XOR_MARK3	"XOR Error "
5	XOR_MARK4	XOR_Error	XOR_MARK4	XOR_MARK4	"XOR Error "
6	XOR_MARK5	XOR_Error	XOR_MARK5	XOR_MARK5	"XOR Error "
7	XOR_MARK6	XOR_Error	XOR_MARK6	XOR_MARK6	"XOR Error "
8	XOR_MARK7	XOR_Error	XOR_MARK7	XOR_MARK7	"XOR Error "
9	XOR_MARK8	XOR_Error	XOR_MARK8	XOR_MARK8	"XOR Error "
10	XOR_MARK9	XOR_Error	XOR_MARK9	XOR_MARK9	"XOR Error "
11	XOR_MARK10	XOR_Error	XOR_MARK10	XOR_MARK10	"XOR Error "
12	XOR_MARK11	XOR_Error	XOR_MARK11	XOR_MARK11	"XOR Error "
13	XOR_MARK12	XOR_Error	XOR_MARK12	XOR_MARK12	"XOR Error "
14	XOR_MARK13	XOR_Error	XOR_MARK13	XOR_MARK13	"XOR Error "
15	XOR_MARK14	XOR_Error	XOR_MARK14	XOR_MARK14	"XOR Error "
16	XOR_MARK15	XOR_Error	XOR_MARK15	XOR_MARK15	"XOR Error "
17	XOR_MARK16	XOR_Error	XOR_MARK16	XOR_MARK16	"XOR Error "
18	XOR_MARK17	XOR_Error	XOR_MARK17	XOR_MARK17	"XOR Error "
19	XOR_MARK18	XOR_Error	XOR_MARK18	XOR_MARK18	"XOR Error "
20	XOR_MARK19	XOR_Error	XOR_MARK19	XOR_MARK19	"XOR Error "
21	XOR_MARK20	XOR_Error	XOR_MARK20	XOR_MARK20	"XOR Error "
22	XOR_MARK21	XOR_Error	XOR_MARK21	XOR_MARK21	"XOR Error "
23	XOR_MARK22	XOR_Error	XOR_MARK22	XOR_MARK22	"XOR Error "
24	XOR_MARK23	XOR_Error	XOR_MARK23	XOR_MARK23	"XOR Error "
25	XOR_MARK24	XOR_Error	XOR_MARK24	XOR_MARK24	"XOR Error "
26	XOR_MARK25	XOR_Error	XOR_MARK25	XOR_MARK25	"XOR Error "

Figure 77: `gcd1_XOR001_mrk_rpt.png`

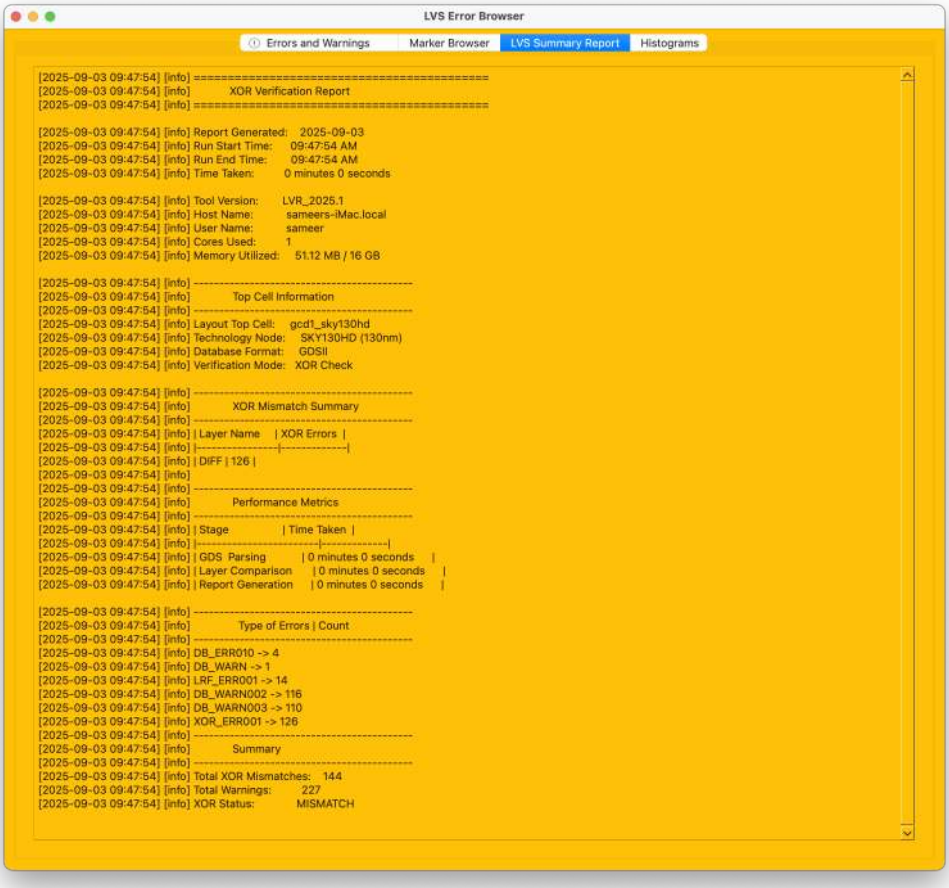


Figure 78: `gcd1_XOR001_sum_rpt.png`

2.18 Test Case: gcd1_XOR002 (XOR)

Purpose and scenario. This example demonstrates a geometric comparison (XOR) failure caused by intentionally *deleting a POLY polygon* in the test layout. The flow compares a *modified* GDS (gcd1_XOR002.gds) against the *golden* reference GDS (gcd1_sky130hd.gds). Any shape that exists in only one of the two layouts appears as an XOR mismatch. In this case, removing a POLY shape creates a non-overlapping region that is flagged by the tool as an XOR error on the POLY layer. The intent is to illustrate how a missing transistor gate/poly feature is caught by XOR even when LVS/DRC might not explicitly call it out.

Figure 79 shows the use-case view highlighting the POLY layer selection and the dense pattern of instances; Figures 80, 81, and 82 capture the error listing, marker table, and summary report produced by the XOR run.

Command file. The complete command file used for this run is reproduced verbatim below.

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db XOR

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd
```

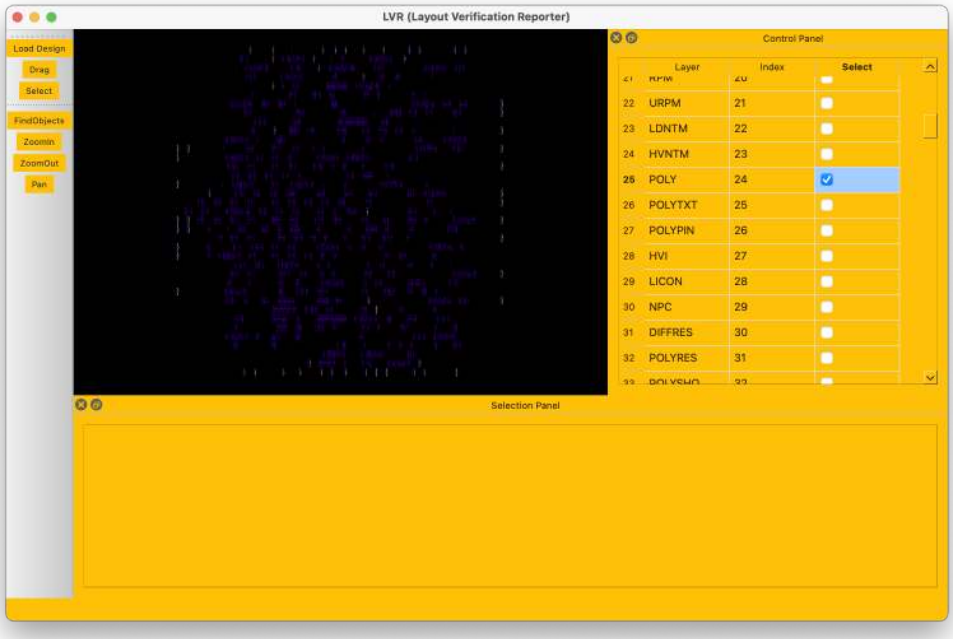


Figure 79: Use case view for **gcd1_XOR002**: POLY layer selected to visualize where the deletion was made.

```
# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_XOR002.gds
read_gds2 ../gds/gcd1_sky130hd.gds
load_db XOR
```

```
no_compare 1

##############################
# XOR settings
##############################
lvs_set_datc_flow 1
#lvs_set_spice_file_name test
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_SVS001A.v
#lvs_set_verilog_netlist2_file_name ../verilog/sky130/gcd1_sky130hd_SVS001B.v
#lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
#lvs_check_open 1
#lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1

# call lvs
##############################
xor_check gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
```

```
set_gui_xor_result 1
#######load_gui
```

What the settings do (how the LVR run is configured).

- `init_tool_db XOR` selects the XOR application mode in LVR, enabling polygon-level comparisons between two layouts.
- `set_technode sky130` loads the technology mapping used for layer names/indices.
- `set_top gcd1_sky130hd` declares the top cell name shared by both layouts; the XOR engine compares shapes under this hierarchy.
- `set_num_cores 1`, `set_die_area`, and `set_num_regions` set runtime and windowing options (single core, full die window here).
- `read_lrf ../rule/sky130.lrf` loads layer/rule formatting so the GUI and reports use consistent layer labels.
- `read_gds ../gds/gcd1_XOR002.gds` loads the *test* (modified) layout; `read_gds2 ../gds/gcd1_sky130hd.gds` loads the *golden* reference.
- `load_db XOR` finalizes database construction for XOR mode; `no_compare 1` disables netlist-based logic (not needed for pure geometric XOR).
- The commented `lvs_*` lines are inert in XOR mode; they are kept for template uniformity but do not affect polygon comparison.
- `xor_check gcd1_sky130hd` runs the actual XOR comparison at the top cell.
- `set_gui_xor_result 1` tells the GUI to open directly on the XOR results for interactive review.

How to run. Place the commands above in a file, for example `gcd1_XOR002.cmd`, then invoke:

```
% LVR gcd1_XOR002.cmd
```

This executes the XOR comparison and produces the error list, marker table, and summary report. Open the GUI (if not auto-opened) to inspect markers over the layout.

Figures and interpretation.

- **Error report** (Fig. 80): lists *XOR_ERR001* entries noting that an XOR mismatch was found for polygons on layer **POLY**. This directly corresponds to the deleted POLY shape in the test layout that no longer overlaps with the reference.
- **Marker browser** (Fig. 81): shows a table of `XOR_MARK*` items, each representing a non-overlapping polygon fragment. The count reflects how many distinct regions of difference exist after Boolean XOR.
- **Summary report** (Fig. 82): aggregates mismatches by layer. Here, the POLY layer shows a large non-zero mismatch count, confirming that the discrepancy is confined to POLY.

2.19 Test Case: `gcd1_XOR003` (XOR)

Use-case overview. This experiment exercises the XOR flow by intentionally shifting a set of `LI` (local interconnect) rectangles in the modified layout and comparing it against the golden reference. The modified database is `gcd1_XOR003.gds` and the golden is `gcd1_sky130hd.gds`. An XOR check geometrically subtracts one layout from the other and reports any polygons that do not cancel. When shapes are translated (even by one grid unit), the overlap disappears and both the “missing” area in one layout and the “extra” area in

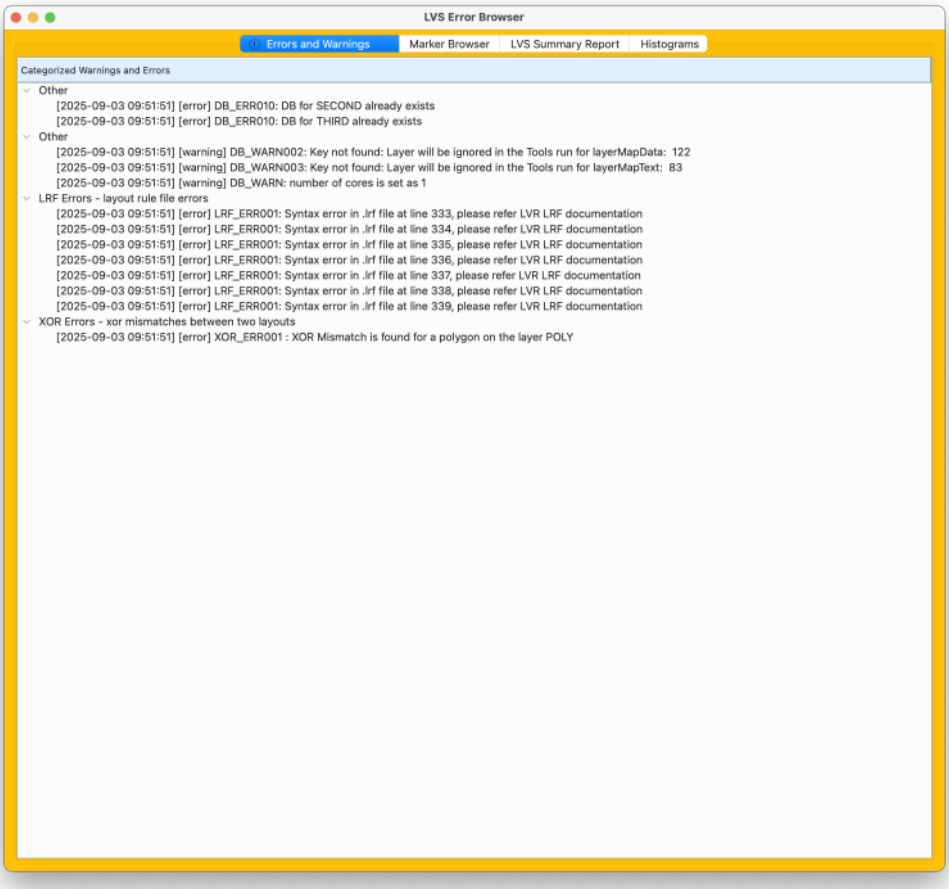


Figure 80: **gcd1_XOR002** error list: XOR mismatches reported on the POLY layer.

the other are flagged as mismatches. Figure 83 shows the *Layout Verification Reporter* with the `LI` layer selected to visualize the shifted rectangles.

Functionally, this scenario represents a common integration failure: a last-minute ECO that moves metal to satisfy congestion or DRC without updating the golden mask set. XOR is ideal here because LVS could still pass (no netlist change), while XOR highlights the mask deltas precisely at polygon level.

Expected results. During the run, the Error Browser will list entries under

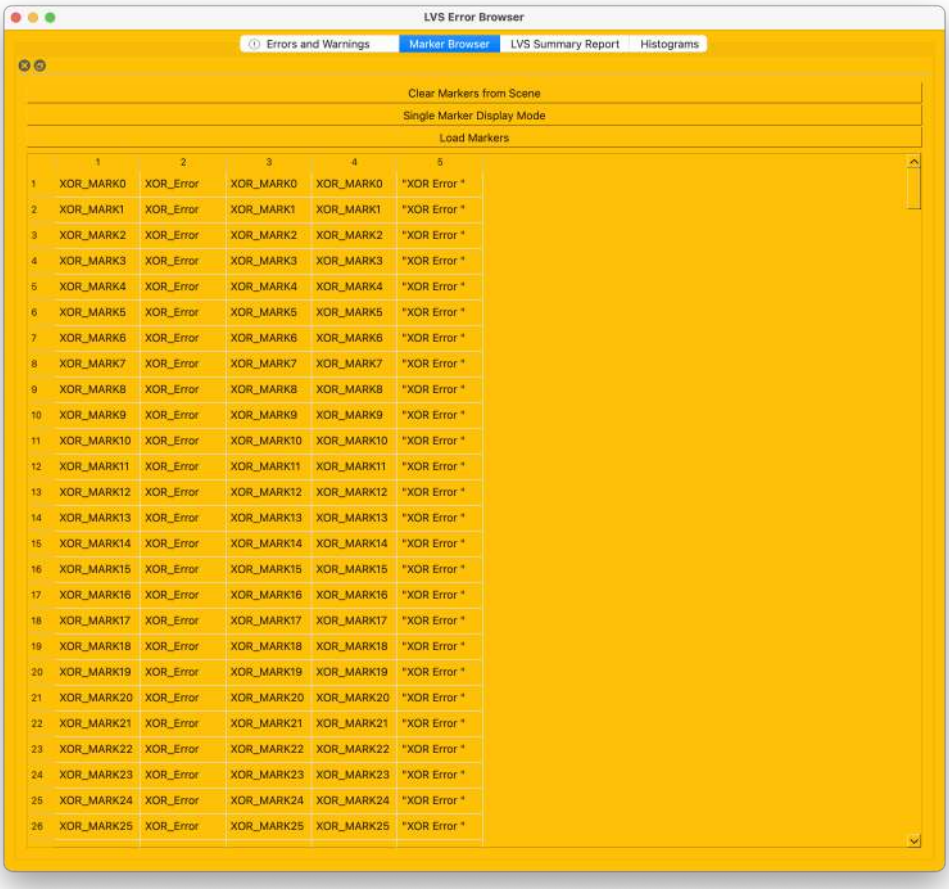


Figure 81: **gcd1_XOR002** marker browser: catalog of XOR_MARK entries (non-overlapping POLY regions).

the *XOR Errors* group (error code `XOR_ERR001`) indicating "XOR mismatch is found for a polygon on the layer LI". The Marker Browser will be populated with many `XOR_MARK*` items—each is a clickable hotspot you can zoom to in LVR. The Summary report will enumerate the count of XOR mismatches for `LI`. See Figures 84–86.

Debug tips. Toggle only `LI` (and optionally `LI-TXT`) in the LVR control panel to reduce clutter; use *FindObjects* on a selected marker to highlight all

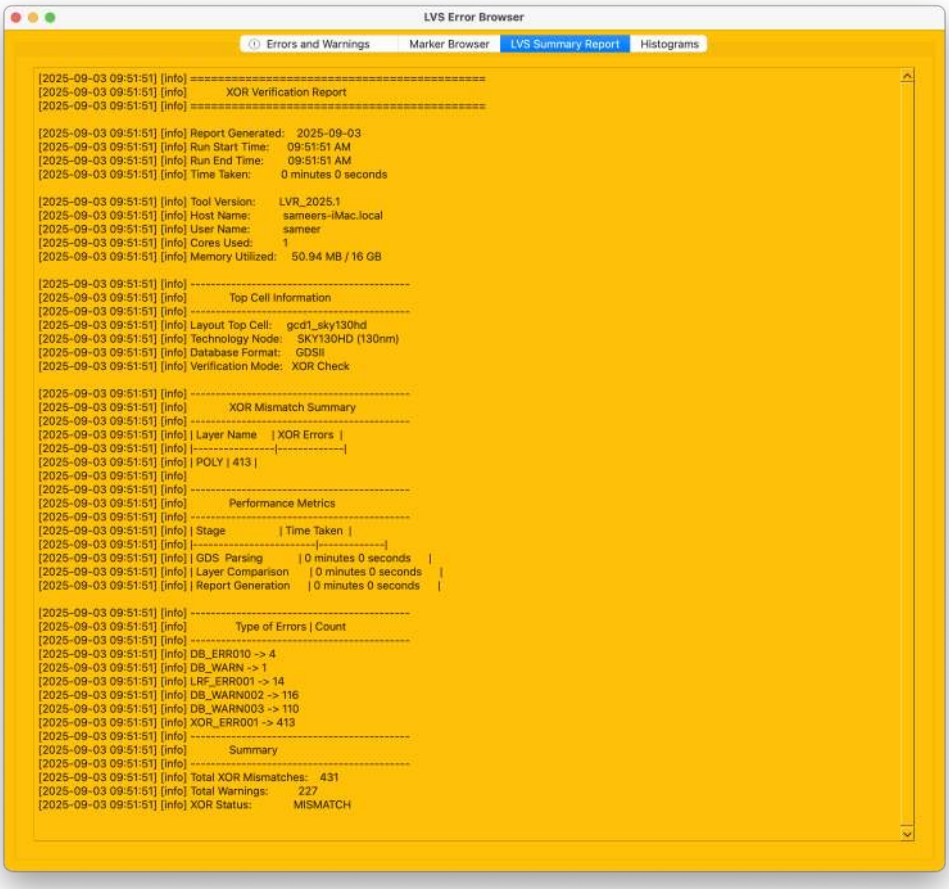


Figure 82: **gcd1_XOR002** summary: mismatch counts by layer, highlighting POLY.

related polygons. Overlaying the two layouts and stepping grid by grid often reveals uniform offsets indicative of a copy/move or alignment mishap.

Command file (verbatim). The following is the exact `gcd1_XOR003.cmd` used to run the check.

```
initiatize tool DB
```

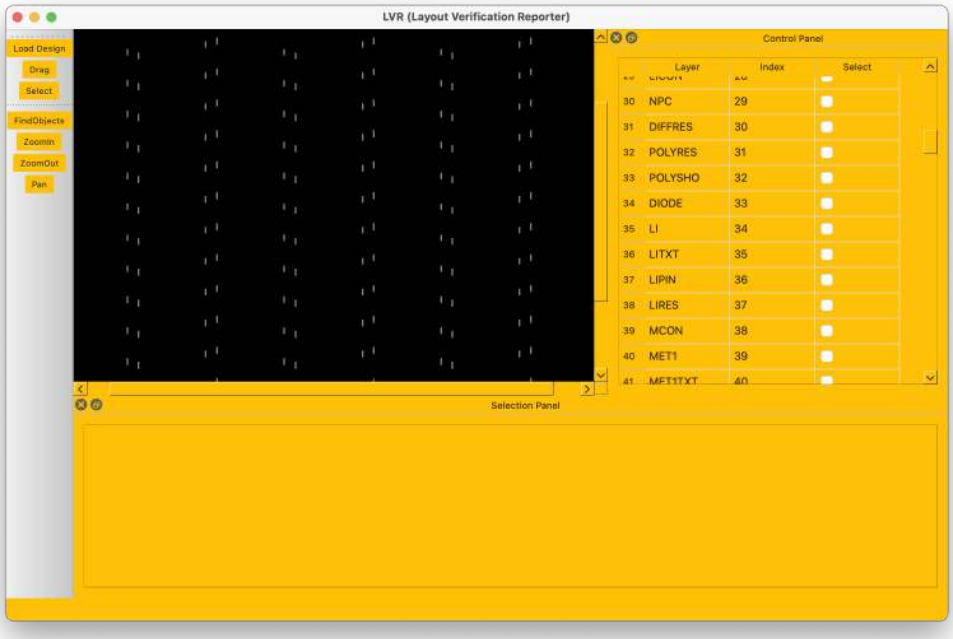


Figure 83: gcd1_XOR003_usecase19.png

```
##############################
#set_debug_mode_3049 0
init_tool_db XOR

set techonlogy

##############################
set_technode sky130

set top module

##############################
set_top gcd1_sky130hd

set number of cores

##############################
set_num_cores 1
set_die_area 0 0 106060 106060
set_num_regions 1
#set_gds_flow 1

read the rule file and gds

##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_XOR003.gds
read_gds2 ../gds/gcd1_sky130hd.gds
load_db XOR
no_compare 1
```

```
##############################

XOR settings

##############################
lvs_set_datc_flow 1
#lvs_set_spice_file_name test
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_SVS001A.v
#lvs_set_verilog_netlist2_file_name ../verilog/sky130/gcd1_sky130hd_SVS001B.v
#lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
#lvs_check_open 1
#lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1

call lvs

##############################
xor_check gcd1_sky130hd

##############################

set and generate report file

##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs
```

```
############################

load gui

############################
#set_gui_xor_result 1
#######load_gui
```

What the settings do (and why).

- `init_tool_db XOR` selects the XOR application mode so the tool interprets subsequent commands in the geometry-diff context rather than LVS/DRC/ERC.
- `set_technode sky130` loads SkyWater 130,nm tech parameters; `read_lrf` brings in layer definitions and mapping from `sky130.lrf`.
- `set_top gcd1_sky130hd` declares the top cell name common to both layouts.
- `read_gds` and `read_gds2` ingest the two GDS databases: the “modified” test case (`gcd1_XOR003.gds`) and the “golden” (`gcd1_sky130hd.gds`).
- `load_db XOR` finalizes DB creation in XOR mode; `no_compare 1` skips any schematic/netlist comparison features (not relevant for pure XOR).
- The commented LVS options (`lvs_*`) are benign here. `lvs_set_datc_flow 1` is harmless in XOR mode and can remain enabled across flows.
- `xor_check gcd1_sky130hd` launches the polygon-level comparison for the named top cell. (It is sufficient to call it once; a second call is optional and redundant.)

- Reporting is viewable in the Error Browser and Summary tabs; explicit `report_lvs` is not required for XOR unless you want an additional textual file.
- GUI: leaving `set_gui_xor_result` commented means you will open the GUI and toggle to XOR results manually; enabling it pre-loads XOR markers.

How to run it. Place the CMD file alongside your `rule/`, `gds/` folders (or adjust relative paths) and invoke:

```
% LVR gcd1_XOR003.cmd
```

When the GUI opens, use the Control Panel to show only the `LI` layer and click markers in the Marker Browser to navigate through mismatches.

About Figure 84. This Error Browser view lists `XOR_ERR001` entries under "XOR Errors — xor mismatches between two layouts". Each item corresponds to a polygonal difference on `LI`, confirming that the shifted rectangles no longer overlap the golden shapes.

About Figure 85. The Marker Browser enumerates all XOR hotspots (`XOR_MARK*`). Selecting a row centers the LVR canvas on that mismatch so you can inspect the offending `LI` polygon(s).

About Figure 86. The Summary Report aggregates counts per layer; here you should see hundreds of mismatches attributed to `LI`, consistent with a systematic shift. Use this count as a quick sanity metric when validating fixes.

Note on histograms. No histogram image was provided for this use-case, so this subsection omits a histogram figure/description.

2.20 Test Case: `gcd1_DEN001` (DENSITY)

Purpose and test idea. This example demonstrates a Density verification failure where the metal density in the bottom region of the layout violates min-

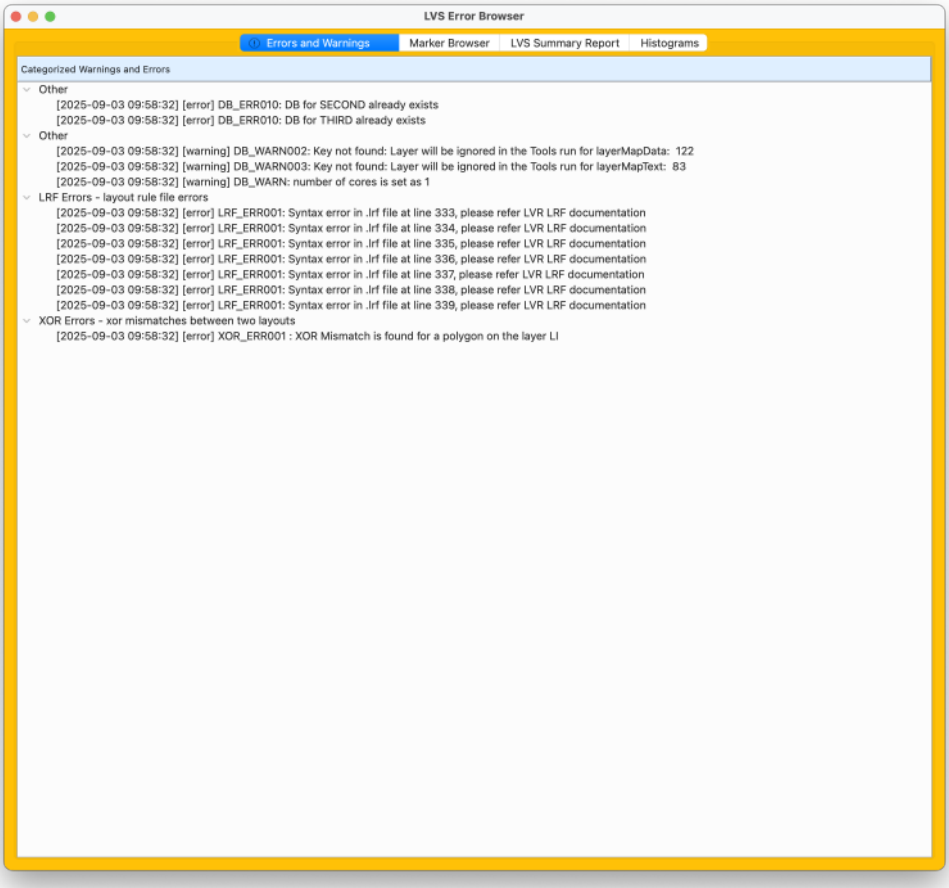


Figure 84: gcd1_XOR003_err_rpt.png

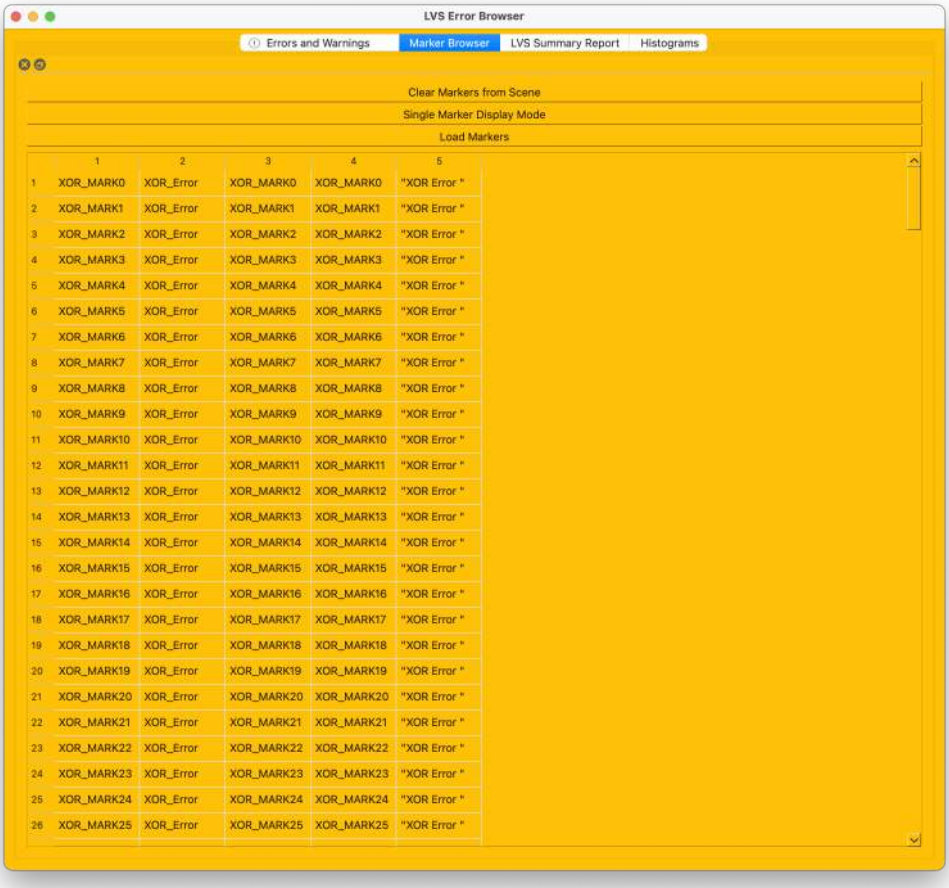


Figure 85: gcd1_XOR003_mrk_rpt.png

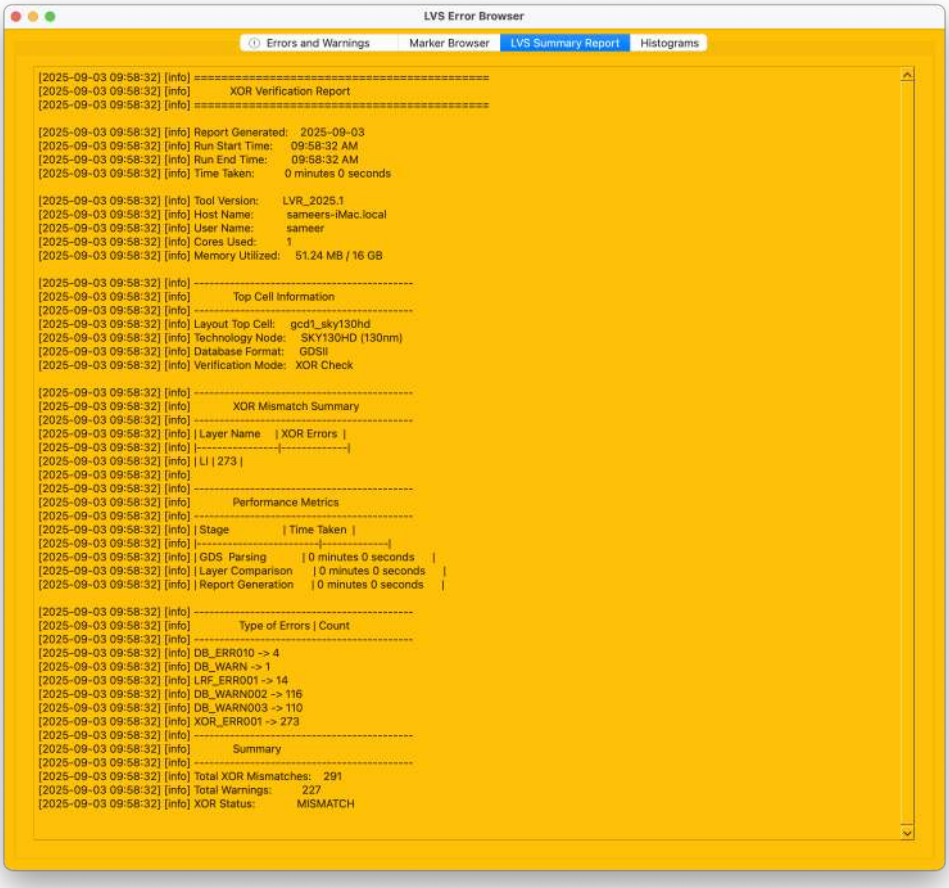


Figure 86: gcd1_XOR003_sum_rpt.png

imum density constraints. The goal of the test `gcd1_DEN001` is to run the Density engine in LVR against the `sky130` technology and report tiles that fall below (or above) configured thresholds. The use-case intentionally leaves a sparse metal pattern near the bottom of the chip area so that the Density checker flags “Min Fail” bins and summarizes them in the GUI and the text report. Figure 87 illustrates the crafted bottom region with intentionally low coverage that triggers density rule violations. During sign-off, such issues are typically addressed by inserting metal fill, adjusting routing, or modifying floorplanning to meet the foundry’s metal density windows.

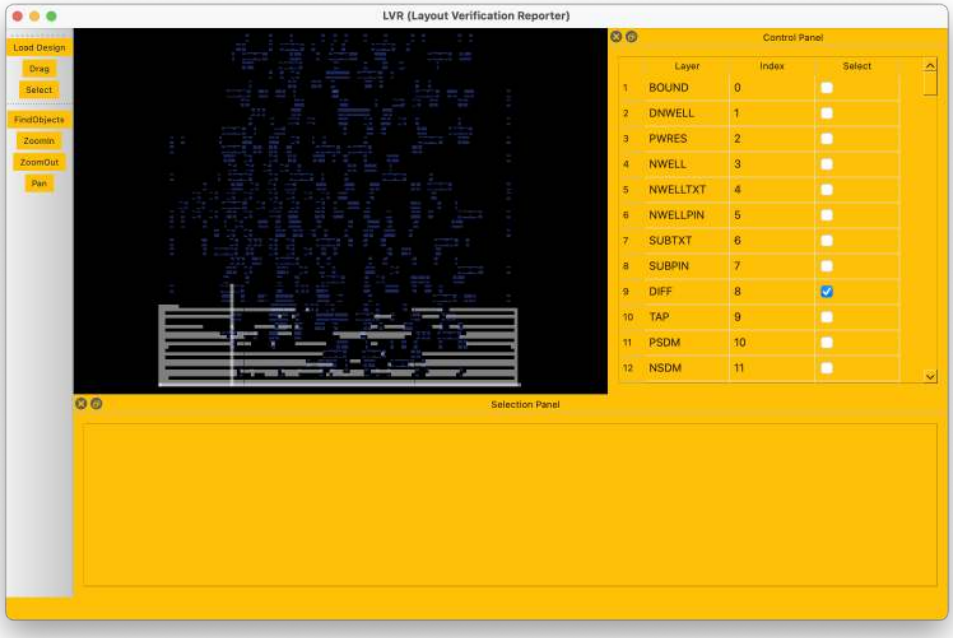


Figure 87: **Use-case layout for** `gcd1_DEN001`: bottom region is intentionally sparse to trigger metal density “Min Fail” errors.

What this test checks. The Density engine tiles the chip area into analysis windows and computes coverage for configured layers (e.g., diffusion, poly,

metals). It then compares measured coverage against minimum/maximum bounds and directional density rules (x/y). Any window out of spec is reported as a marker and summarized in the run report and histograms.

Inputs and flow. The flow initializes the Density tool database, selects the `sky130` technology, loads the golden GDS layout, and invokes `density` on the top cell `gcd1_sky130hd`. Multicore and multi-region settings are enabled to parallelize the computation. Reports are produced in the LVR Error Browser (Density mode) with tabs for Errors & Warnings, Marker Browser, Density Summary Report, and Histograms.

Command file: `gcd1_DEN001.cmd` **(verbatim)**

```
# initiatize tool DB
##############################
#set_debug_mode_3049 0
init_tool_db DENSITY

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 4
set_num_regions 4
#set_gds_flow 1
```

```
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../gds/gcd1_sky130hd.gds
#read_gds ../gds/gcd1_DEN001.gds
load_db DENSITY
no_compare 1

##############################
# DENSITY settings
##############################
lvs_set_datc_flow 1
#lvs_set_spice_file_name test
#lvs_set_verilog_netlist_file_name ../verilog/sky130/gcd1_sky130hd_SVS001A.v
#lvs_set_verilog_netlist2_file_name ../verilog/sky130/gcd1_sky130hd_SVS001B.v
#lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
#lvs_check_open 1
#lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1

# call lvs
##############################
density gcd1_sky130hd

##############################
# set and generate report file
##############################
```

```
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
#load_gui
```

Settings explained (what the command file does)

- `init_tool_db DENSITY` selects the Density verification context.
- `set_technode sky130` loads foundry rule semantics from `sky130.lrf`.
- `set_top gcd1_sky130hd` defines the layout's top cell.
- `set_num_cores 4` and `set_num_regions 4` enable parallel tiling and evaluation—useful for large designs.
- `read_lrf ../rule/sky130.lrf` points to the layout rule file; density bounds and window sizes are derived from here.
- `read_gds ../gds/gcd1_sky130hd.gds` loads the reference layout to be checked (the edited test GDS is commented out in this run).
- `load_db DENSITY` builds the LVR database in Density mode; `no_compare 1` disables netlist comparison since density is geometry-only.
- `lvs_set_datc_flow 1` keeps a unified backend flow; it is harmless here and allows consistent job setup across engines.
- `density gcd1_sky130hd` runs the Density checker over the full die, generating Errors/Markers/Summary/Histogram outputs.
- Report generation and GUI load are optional (commented). The Density Error Browser will still be populated for interactive review.

How to run

Invoke the tool with the command file:

```
%LVR gcd1_DEN001.cmd
```

After completion, open the Density Error Browser to inspect Errors & Warnings, markers, the summary, and histograms.

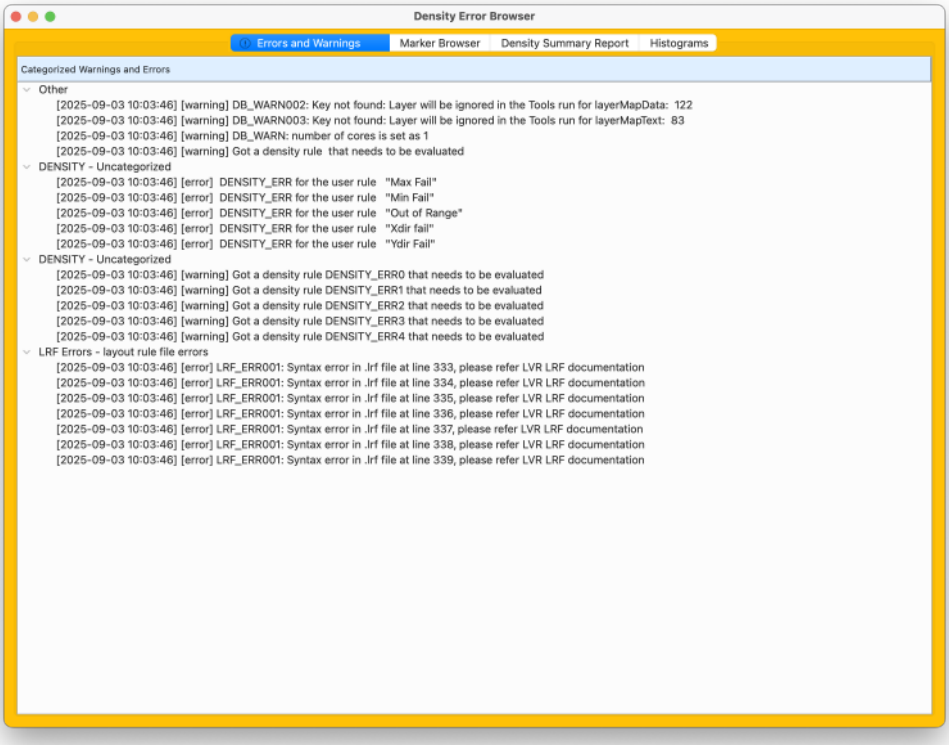


Figure 88: **Errors & Warnings** for `gcd1_DEN001`. Shows density rule categories received (Min/Max/OutOfRange/X/Y directional) and any LRF parser notices.

About Fig. 88. This pane lists Density rule evaluations and LRF notices. In this case, Density errors are registered for “Min Fail” (low coverage) along with

informational warnings from rule parsing. These entries corroborate that the bottom region violates minimum metal density.

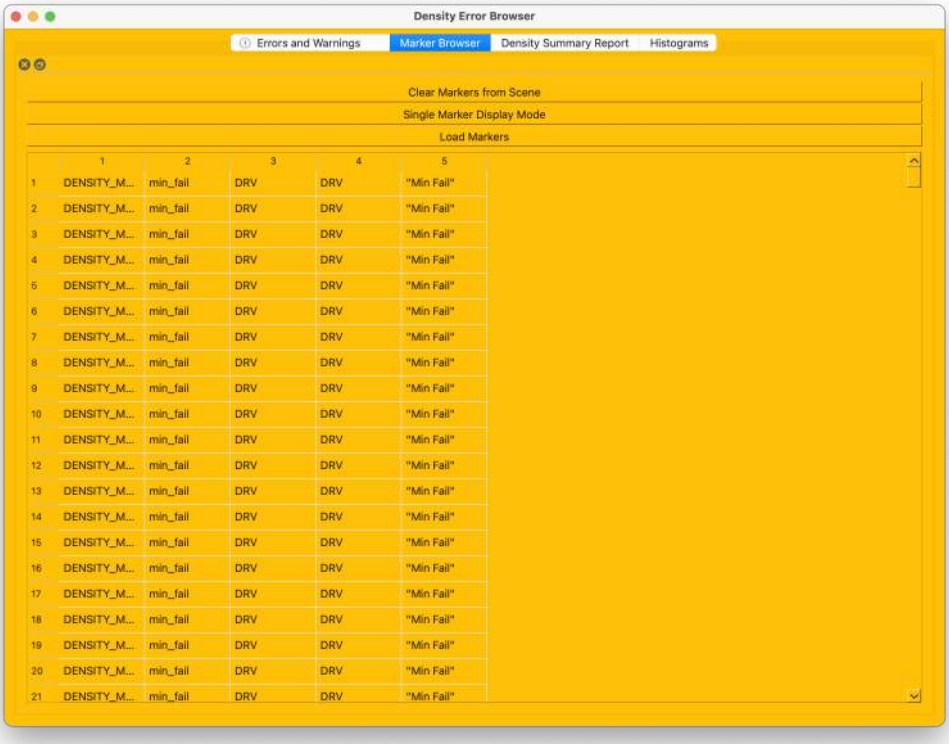


Figure 89: **Marker Browser** for `gcd1_DEN001`. Each row is a density-window violation (e.g., “Min Fail”) and can be navigated to in the viewer.

About Fig. 89. The Marker Browser enumerates all failing tiles. Double-clicking a row centers the viewer on that window, allowing quick inspection of the sparse region that triggered the “Min Fail” classification.

About Fig. 90. This report summarizes violation counts per rule category and shows job details (technology, database, verification mode) plus timing.

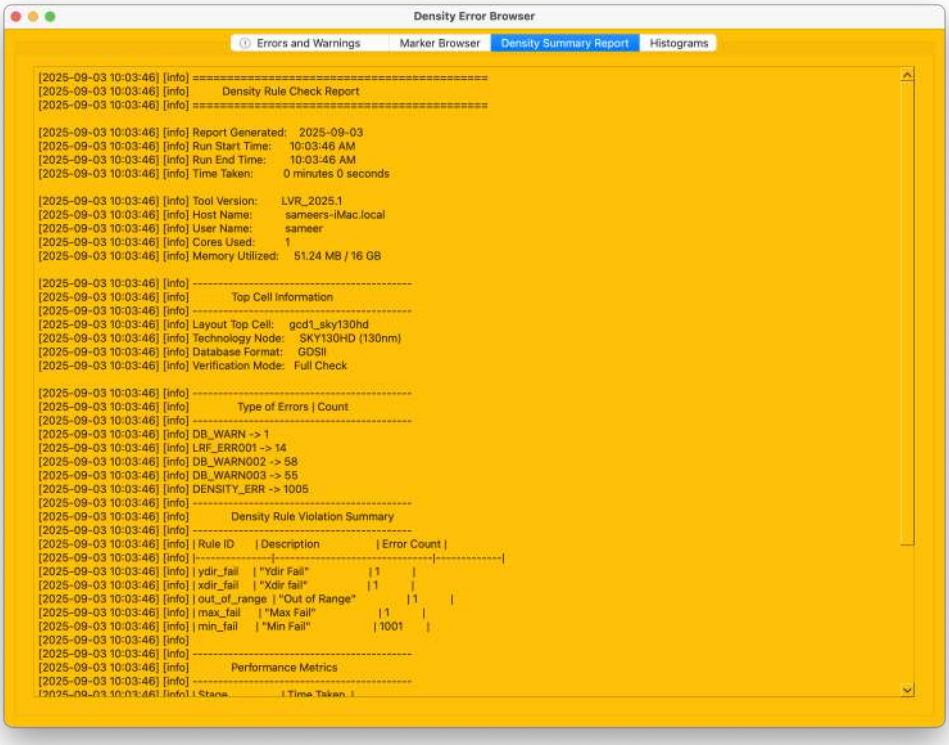


Figure 90: **Density Summary Report** for `gcd1_DEN001`. Aggregates counts per rule ID (min_fail, max_fail, out_of_range, xdir_fail, ydir_fail) and records performance metrics.

For this case, *min_fail* dominates, consistent with intentionally low density at the bottom.

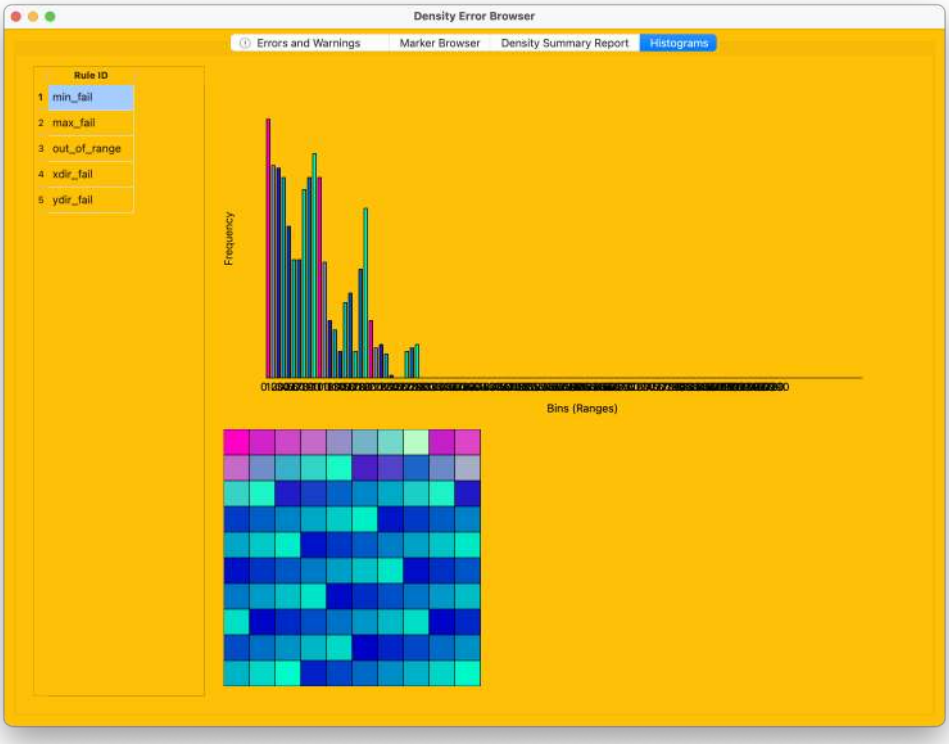


Figure 91: **Histograms** for `gcd1_DEN001`. Bar charts and heatmap visualize the distribution of failing bins across the die and rule categories.

About Fig. 91. The histogram view shows frequency per bin range and a spatial heatmap of violations. Concentration in the lower rows aligns with the crafted sparse bottom region of Fig. 87.

2.21 Test Case: gcd1_MLVS005 (MLVS)

Purpose and scenario. This test demonstrates a *macro-level* Layout-vs-Schematic (MLVS) run on a single standard cell (macro) rather than the full chip. The goal is to validate that the layout of the macro in the GDS matches its schematic view (SPICE/CDL), with macro pins handled explicitly by the flow. In this example, we verify the Sky130 HD library cell sky130_fd_sc_hd__clkinv_16 wrapped by a tiny "TOP" design. Figure 92 shows the region of interest in the LVR viewer, where LI and MET1 geometry and pin text layers are toggled to visualize device/pin placement and connectivity.

At a high level, the run: (i) initializes the tool DB in MLVS mode, (ii) loads the Sky130 LRF rule file and the macro-level GDS, (iii) points MLVS to the macro's SPICE netlist and the technology CDL (for primitive references), (iv) disables macro-pin overriding and explicit port enables (so the macro's own pin semantics are used), and (v) executes mlvs TOP to compare the extracted macro layout against its schematic. The summary report indicates child-cell coverage and pass/fail status for each macro instance.

What this test checks. Unlike full-chip LVS, MLVS focuses on matching *macro cells*. It reports (1) top-cell match, (2) list of child cells (macro names), (3) coverage statistics (how many instances were verified), and (4) per-macro pass/fail. Common pitfalls caught here include missing device fingers, wrong pin text, misaligned contacts, or incorrect layer usage internal to the macro. For this run, the MLVS Summary Report (Fig. 95) shows that sky130_fd_sc_hd__clkinv_16 instances are verified and pass, with zero shorts/opens shown in the histograms tab (Fig. 96).

Command file gcd1_MLVS005.cmd **(verbatim).**

```
# initiatize tool DB
##############################
```

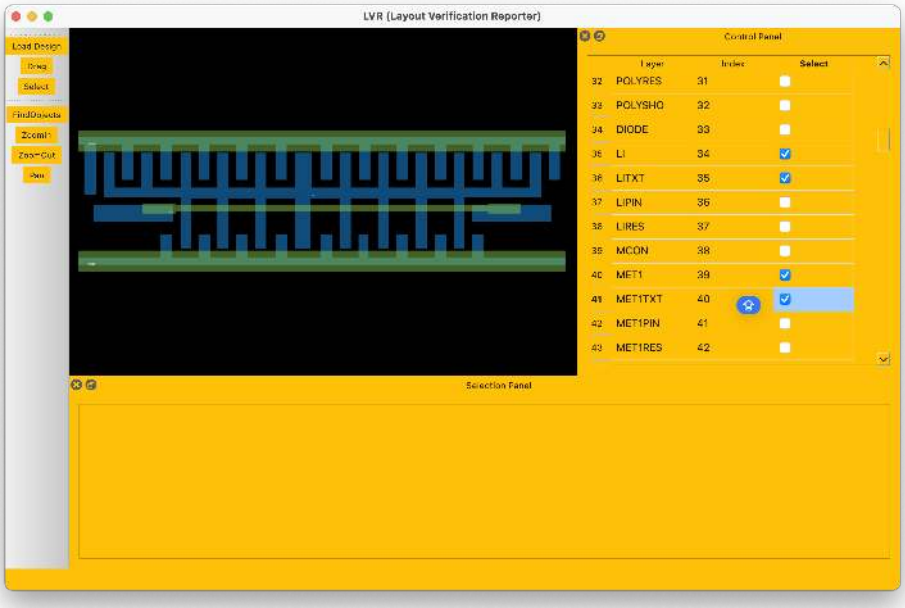


Figure 92: **Use-case view** for `gcd1_MLVS005`: Macro geometry and pin text (e.g., LI, MET1, MET1TXT) are enabled to inspect the standard-cell layout used by MLVS.

```
init_tool_db MLVS

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top TOP

# set number of cores
##############################
set_num_cores 1
set_num_regions 1
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130.lrf
read_gds ../macro_gds/sky130_fd_sc_hd__clkinv_16_top.gds
load_db MLVS
no_compare 1

##############################
# MLVS settings
##############################
lvs_set_datc_flow 0
lvs_set_spice_file_name ../macro_tops/sky130_fd_sc_hd__clkinv_16_top.sp
lvs_set_macros_cdl_file_name ../spice/sky130.cdl
#lvs_enable_ports 1
#lvs_check_shorts 1
```

```
#lvs_check_open 1
#lvs_check_macro 0
#lvs_check_matching 1
#lvs_check_unconnected_pins 1
lvs_enable_ports 0
lvs_consider_macro_pins 0

# call lvs
##############################
mlvs TOP

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
load_gui
```

Settings explained (one-page overview). `init_tool_db MLVS` selects macro LVS mode, which activates macro-aware extraction and reporting. `set_technode sky130` loads process defaults, naming, and device decks expected by the Sky130 rule file. The design under test is `TOP` (`set_top TOP`), a tiny wrapper that instantiates the target macro. The Sky130 LVR rule file is brought in via `read_lrf ../rule/sky130.lrf`, followed by the macro GDS: `read_gds ../macro_gds/sky130_fd` The database is then bound to the MLVS context with `load_db MLVS`. With `no_compare 1` set earlier in many flows to defer checks, the actual compari-

son is explicitly triggered later by `mlvs TOP`.

Under “MLVS settings,” `lvs_set_datc_flow 0` disables the DATC Verilog path since we are comparing GDS vs. SPICE/CDL at macro granularity. The schematic view for the top wrapper is given by `lvs_set_spice_file_name ../macro_tops/sky130_` Primitive devices referenced by the macro are resolved using the technology CDL: `lvs_set_macros_cdl_file_name ../spice/sky130.cdl`. We keep `lvs_enable_ports 0` so ports are not force-enabled globally (the macro’s pin text layers and the wrapper netlist govern connectivity), and `lvs_consider_macro_pins 0` avoids pin promotion across hierarchy—useful when the macro already carries correct pin text and shapes internally.

Finally, `mlvs TOP` runs the macro LVS. Reports can be dumped with `report_lvs` (optional), and the GUI is launched via `load_gui` for visual diagnosis of any cell mismatches, shorts/opens, or pin text issues.

How to run. Place the above commands in `gcd1_MLVS005.cmd` and invoke:

```
%LVR  gcd1_MLVS005.cmd
```

The tool will parse the LRF, load the macro GDS and netlists/CDL, perform macro-level LVS on `TOP`, and generate interactive reports.

About Fig. 93. This pane groups database warnings (e.g., layer-map keys) and LRF syntax messages separately from MLVS comparison results. For this test, no macro-level mismatches are reported—useful confirmation that the macro geometry and pins are consistent with the schematic.

About Fig. 94. The marker table would enumerate each mismatch location (type, instance, pin/net, and coordinates). An empty list implies no device/pin discrepancies at macro level.

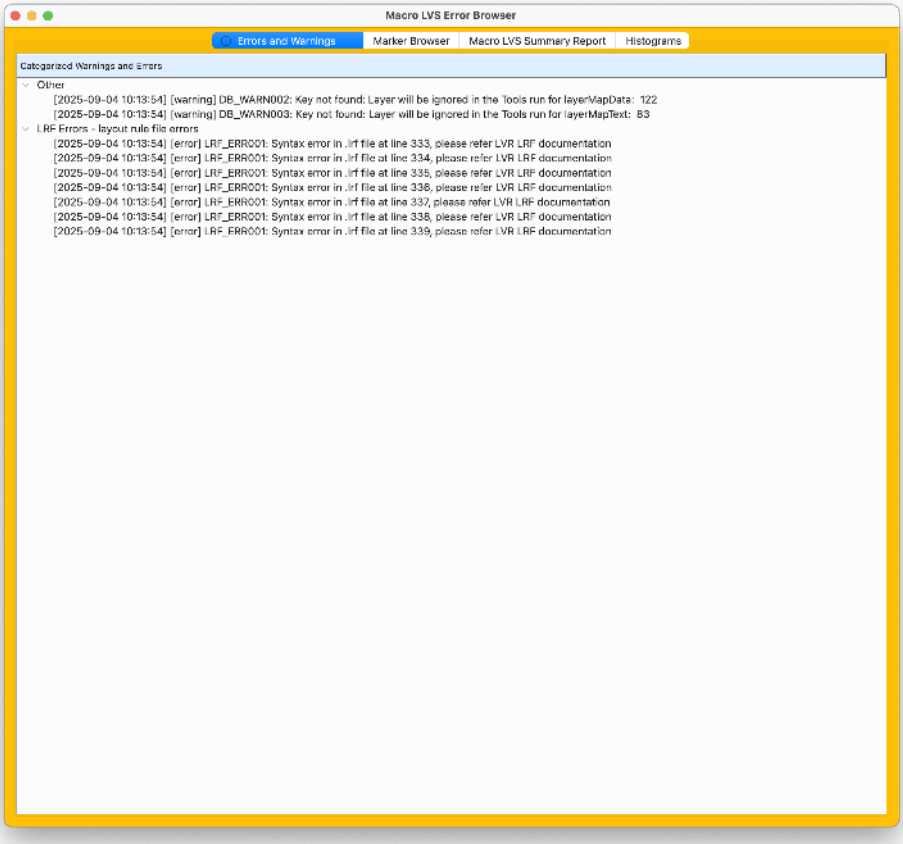


Figure 93: **Errors and Warnings** for `gcd1_MLVS005`. LRF parse notes and general DB warnings are listed first; macro LVS-specific errors would appear here if any.

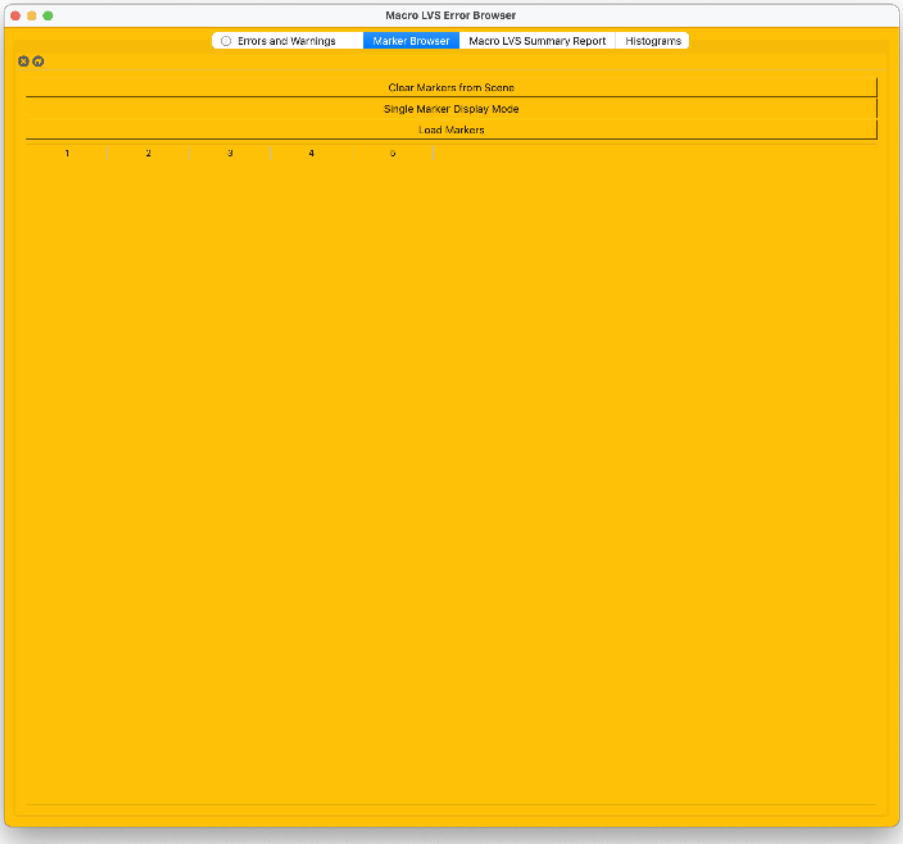


Figure 94: **Marker Browser** for `gcd1_MLVS005`. No MLVS error markers are present, indicating clean macro comparison.

Figure 95: **MLVS Summary Report**. Top-cell match, child-cell coverage, and per-macro pass/fail. Here `sky130_fd_sc_hd__clkinv_16` instances are verified and *PASS*.

About Fig. 95. This is the primary “at-a-glance” result for macro verification—counts of layout vs. schematic occurrences for each child cell and the final result.

Figure 96: **Histograms**. No shorts/opens/unconnected-pin distributions are visible, consistent with a clean MLVS.

2.22 Test Case: `gcd1_DRC001` (DRC)

Context. From its name, `gcd1_DRC001` is a **DRC** testcase (Design Rule Check). The purpose is to validate that the LI (local interconnect) polygons in the

gcd1_sky130hd layout meet the minimum *area* requirement. This testcase intentionally injects a failing area threshold in the rule file to demonstrate how the tool reports and localizes such violations.

Use case and intent. Figure 97 illustrates the portion of the layout used for this check: repeated LI geometries in a regular pattern that make area errors easy to spot at multiple sites. The scenario exercises the rule "*li.6 : min. li area*" and shows how the DRC engine flags polygons whose area falls below the configured minimum.

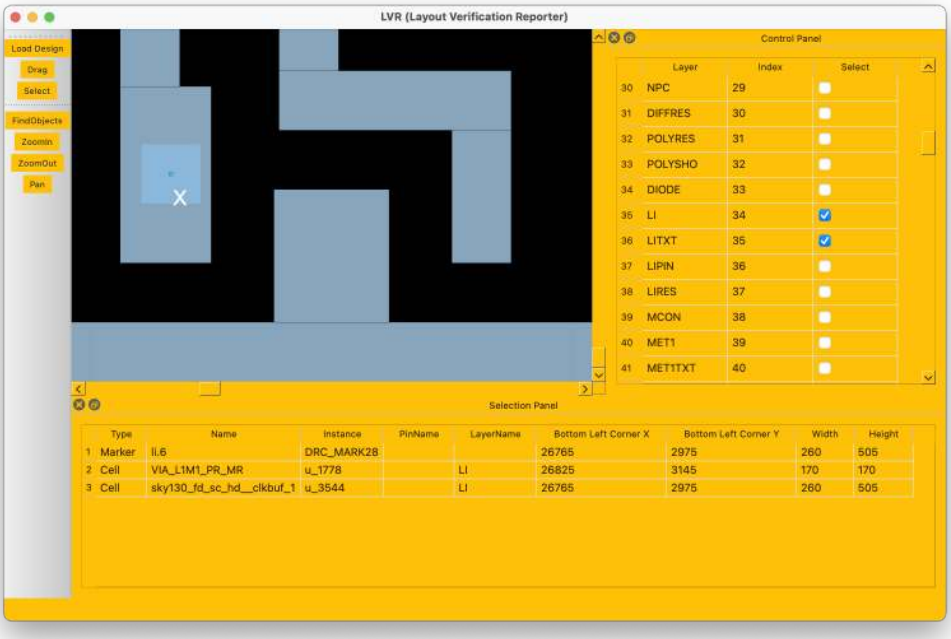


Figure 97: **gcd1_DRC001 use case.** Layout window highlighting LI shapes used to trigger the minimum-area rule.

Rule configuration (error injection). To make the behavior explicit, the rule deck is edited so that the LI area limit is enforced in a way that will intentionally

fail small tiles. Figure 98 shows the relevant lines in the rule deck, where the `li.6` check expresses the minimum *area* for LI polygons. In this testcase, the effective threshold is set so that many LI islands violate it, producing numerous markers that are easy to review in the DRC browsers.

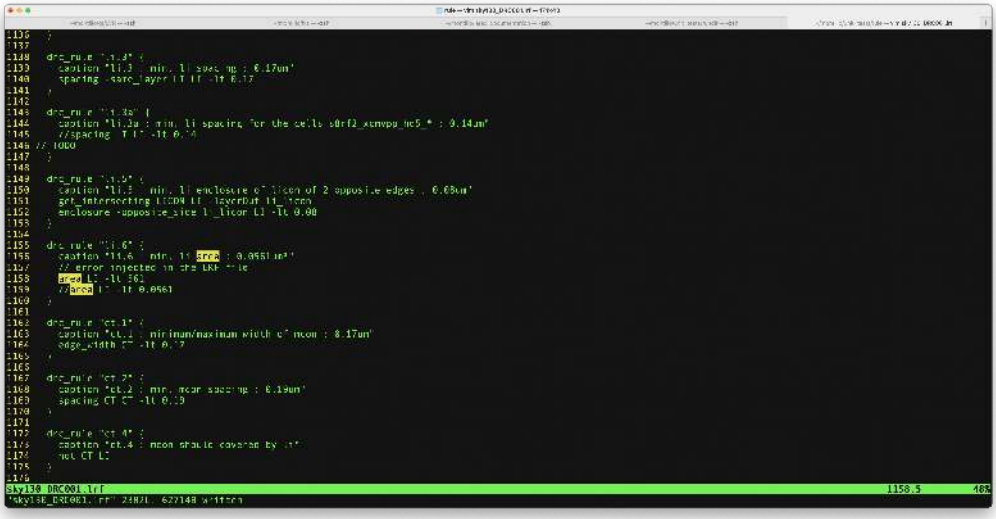


Figure 98: **Rule file excerpt for error injection.** The `li.6` minimum LI area rule is emphasized to intentionally create violations for demonstration.

What to look for in the reports. After running DRC, the Errors & Warnings view (Figure 99) lists the rule IDs and messages for each failure instance. The Marker Browser (Figure 100) enumerates one row per violation, which you can double-click to center in the layout. The Summary (Figure 101) aggregates counts by rule and records environment details (tool version, node, hierarchy mode). Finally, the Histogram view (Figure 102) gives a quick sense of distribution across placement bins and a heat-map for density of markers.

Command file (verbatim).

```
# initiatize tool DB
##############################
```

```
init_tool_db DRC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130_DRC001.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db DRC
no_compare 1

##############################
# DRC settings
##############################
lvs_set_datc_flow 1

# call lvs
##############################
drc gcd1_sky130hd
```

```
############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

What these settings do (about one page). This testcase initializes the tool in **DRC** mode via `init_tool_db DRC`, which activates the rule-checking pipeline rather than comparison flows (e.g., LVS/XOR). The technology is set to `sky130` so the proper layer map and unit system are used. The top cell is `gcd1_sky130hd`; providing it early lets the tool pre-allocate resources and reference it consistently in later steps.

The run is single-threaded here (`set_num_cores 1`) for reproducibility; multi-core can be used for throughput. An explicit die area is registered (`set_die_area 0 0 106060 10606` The rule file is then loaded with `read_lrf ../rule/sky130_DRC001.lrf`; this deck contains the edited *li.6* minimum-area rule (see Figure 98). The design geometry is provided through `read_gds ../gds/gcd1_sky130hd.gds` and bound to the database with `load_db DRC`.

`no_compare 1` disables any schematic/layout equivalence work so only pure rule checking runs. Although labeled “lvs_set_*”, `lvs_set_datc_flow 1` is a benign global toggle in this environment to use a streamlined data/tiling path for performance; it does not convert this into an LVS run.

The actual DRC is launched by `drc gcd1_sky130hd`. On completion, opening the GUI (`load_gui`) provides four coordinated views: (i) *Errors &*

Warnings to see rule text and counts, (ii) *Marker Browser* to list each violation and navigate to it, (iii) *DRC Summary Report* with environment metadata and per-rule totals, and (iv) *Histograms* to visualize spatial distribution across bins and a heat map. This test intentionally yields a large set of `li.6` markers so reviewers can practice triage, filtering, and "zoom-to-marker" workflows.

How to run. Place the command file in your run directory and execute:

```
% LVR <cmd_file.cmd>
```

For example: `% LVR gcd1_DRC001.cmd`. Results populate the report panes automatically; you can export summaries from the GUI if needed.
Description. Figure 99 shows the log categorization and the DRC section listing `li.6` failures. Each entry names the rule and provides a concise description (*min. li area*) to aid filtering.
Description. In Figure 100, the Marker Browser lists every hit for `li.6`. Double-clicking a row centers that polygon in the canvas so you can measure area and confirm the violation.
Description. Figure 101 aggregates totals, confirms node (`SKY130HD`), database format (GDSII), verification mode, and the number of `li.6` errors produced by the injected threshold.
Description. Figure 102 visualizes how the `li.6` area violations cluster; peaks indicate repeated small LI tiles, while the heat-map highlights regions with the highest concentration of fails.

2.23 Test Case: `gcd1_DRC002` (DRC)

Intent and use-case. This testcase demonstrates how the DRC engine flags a *minimum width* violation on the POLY layer. The GDS `gcd1_sky130hd.gds` intentionally contains narrow poly fingers so that rule `poly.1a : min. poly width : 0.15um` is violated in many locations. The goal is to show end-

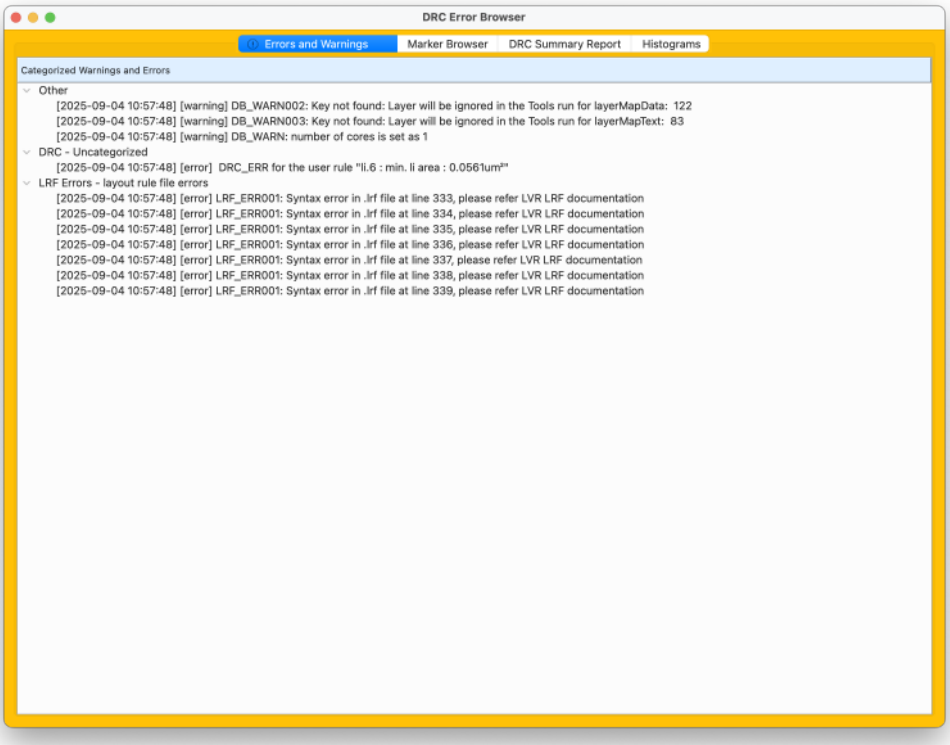


Figure 99: **Errors & Warnings.** Extract of rule messages for `li.6` minimum LI area violations.

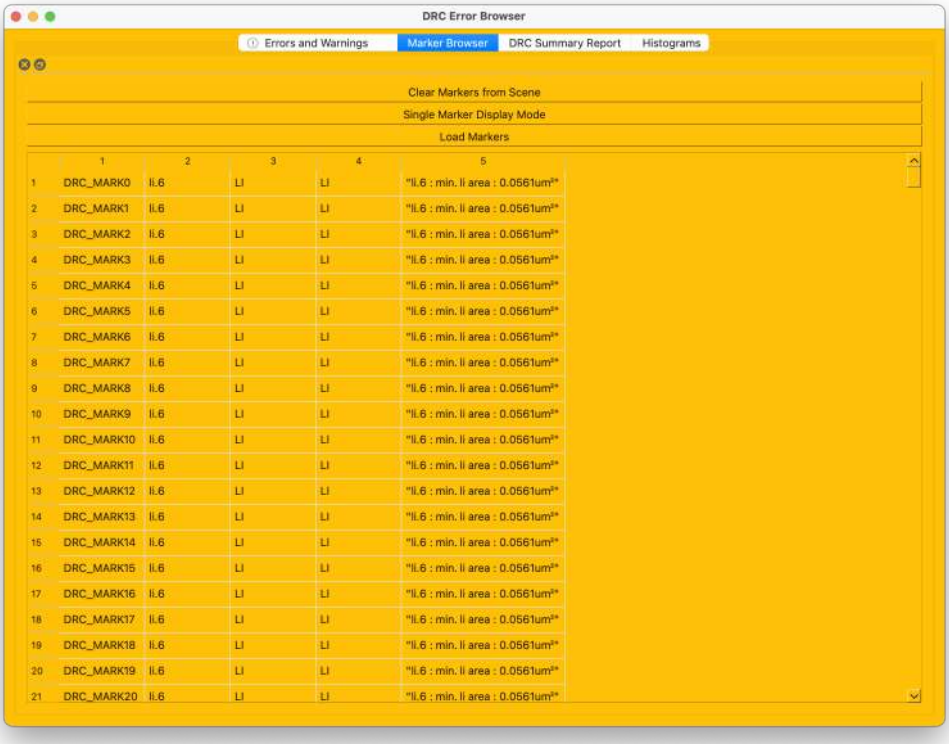


Figure 100: **Marker Browser.** One row per violation with rule ID, layer, and a short message.

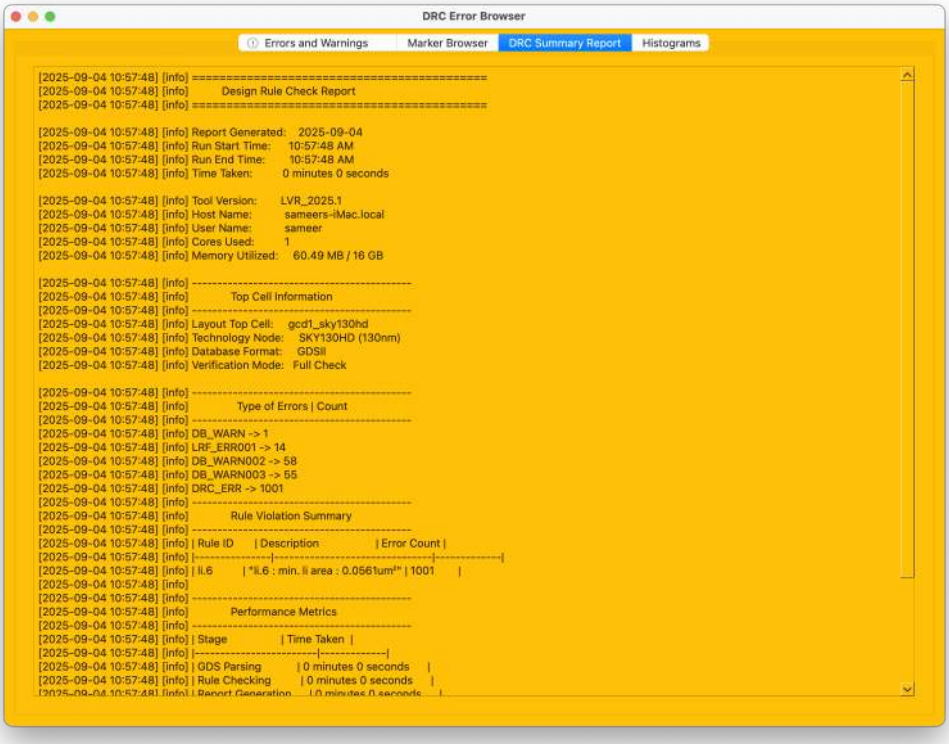


Figure 101: **DRC Summary Report.** Environment details and per-rule counts for this run.

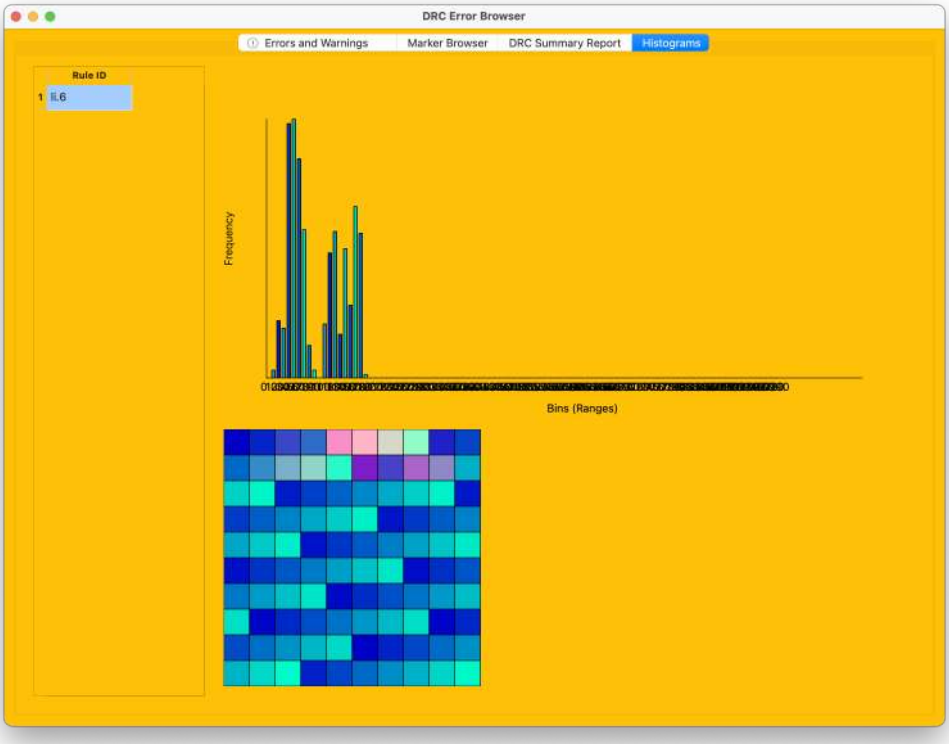


Figure 102: **Histograms.** Distribution of `li.6` markers over spatial bins with an error heat-map.

to-end flow: loading the DRC deck, running checks, browsing markers, and interpreting the summary/histogram outputs.

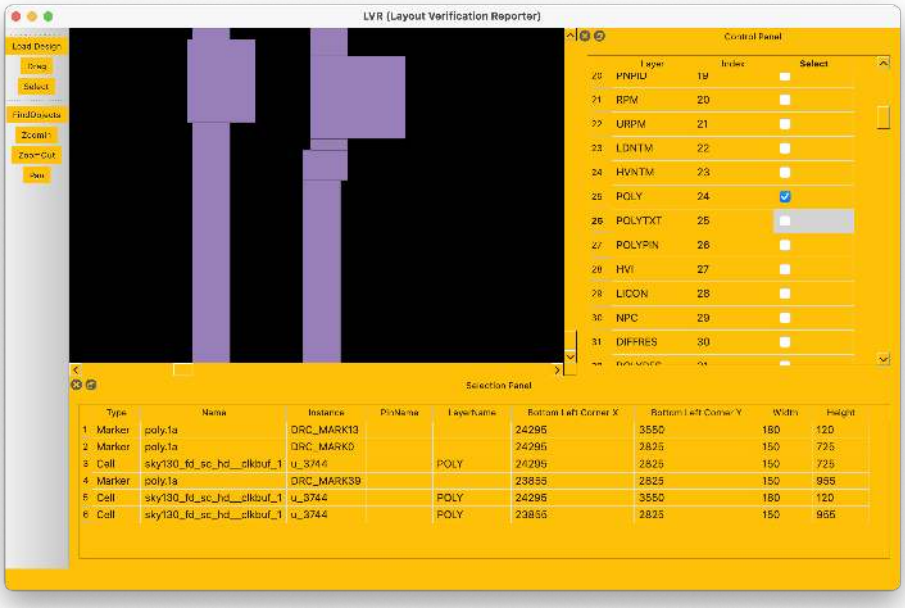


Figure 103: gcd1_DRC002_usecase23.png

Figure 103 shows representative geometry used to stress the POLY width rule: tall, slender polysilicon shapes and necked regions. These intentionally undersized features guarantee that the check trips in a dense, realistic context.

Rule deck (error insertion) for this testcase. The rule file used for this testcase (`../rule/sky130_DRC002.lrf`) purposely tightens the POLY minimum width so violations are guaranteed. The snippet in Figure 104 highlights the injected setting.

In Figure 104, rule `poly.1a` (*min. poly width*) is shown with the correct threshold commented and a stricter/easier-to-violate value enabled. This “error injection” ensures the run produces ample hits for documentation and training.

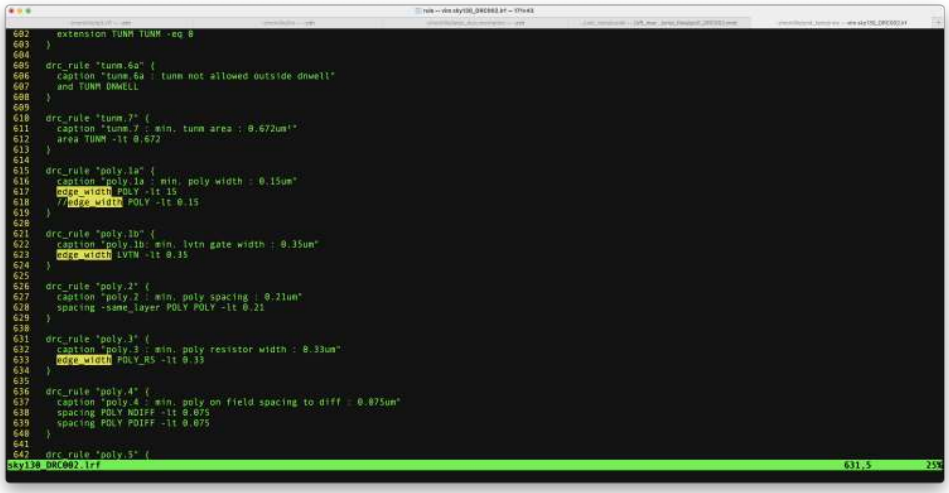


Figure 104: gcd1_DRC002_err_inj.png

Command file used to run the testcase. The following is the verbatim content of `gcd1_DRC002.cmd`; it drives the DRC flow for this test.

```
# initiatize tool DB
##############################
init_tool_db DRC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
```

```
set_num_cores 1
set_die_area 0 0 106060 106060
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130_DRC002.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db DRC
no_compare 1

##############################
# DRC settings
##############################
lvs_set_datc_flow 1

# call lvs
##############################
drc gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
# load gui
##############################
load_gui
```

What the settings do (how LVR/DRC is configured). `init_tool_db DRC` selects the DRC application mode. `set_technode sky130` loads technology assumptions and layer map for SKY130. The `set_top` specifies the top cell name in the layout database. `set_num_cores 1` fixes the run to a single CPU thread, and `set_die_area` constrains the design window (useful for tiling or region runs). The core inputs are the rule deck and the layout: `read_lrf ../rule/sky130_DRC002.lrf` brings in the DRC rules (including the injected min-width value), while `read_gds ../gds/gcd1_sky130hd.gds` supplies the test GDS. `load_db DRC` compiles these into the runtime database; `no_compare 1` confirms this is a pure DRC run (no schematic compare). `lvs_set_datc_flow 1` enables standard infrastructure used across flows. The actual check is launched by `drc gcd1_sky130hd`. Optional report lines are commented (the GUI already renders summaries). Finally, `load_gui` opens the DRC Error Browser to inspect results.

How to run. From a shell in the directory containing the command file, run:

```
% LVR gcd1_DRC002.cmd
```

This executes the flow, builds the database, performs DRC, and opens the GUI to review errors, markers, and histograms.

Reports and interpretation. Figure 105 (Errors & Warnings view) shows the fired rule `poly.1a : min. poly width : 0.15um`. It lists parser notes and categorizes issues; the key takeaway is that the POLY width rule is violated extensively, as intended.

Figure 106 (Marker Browser) enumerates individual hits (e.g., `DRC_MARK0`, `DRC_MARK1`, ...), each labeled with the same rule text. Selecting a row centers the viewer on the violating polygon so its local context (neighbor layers, necking, etc.) can be examined.

Figure 107 is the DRC Summary Report. It captures top-cell metadata, technology, verification mode, and a Rule Violation Summary. The row for

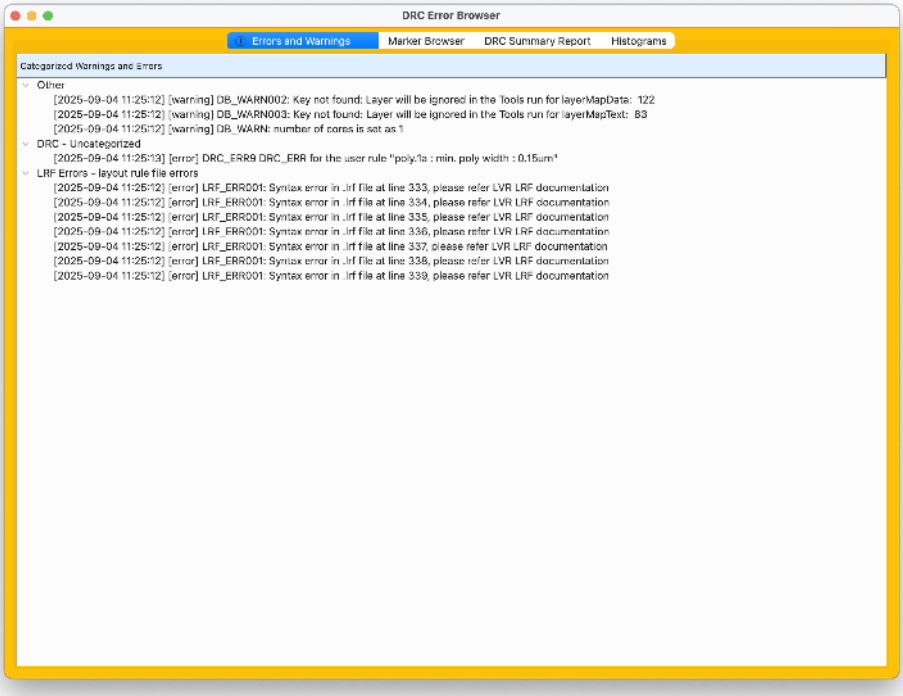


Figure 105: gcd1_DRC002_err_rpt.png

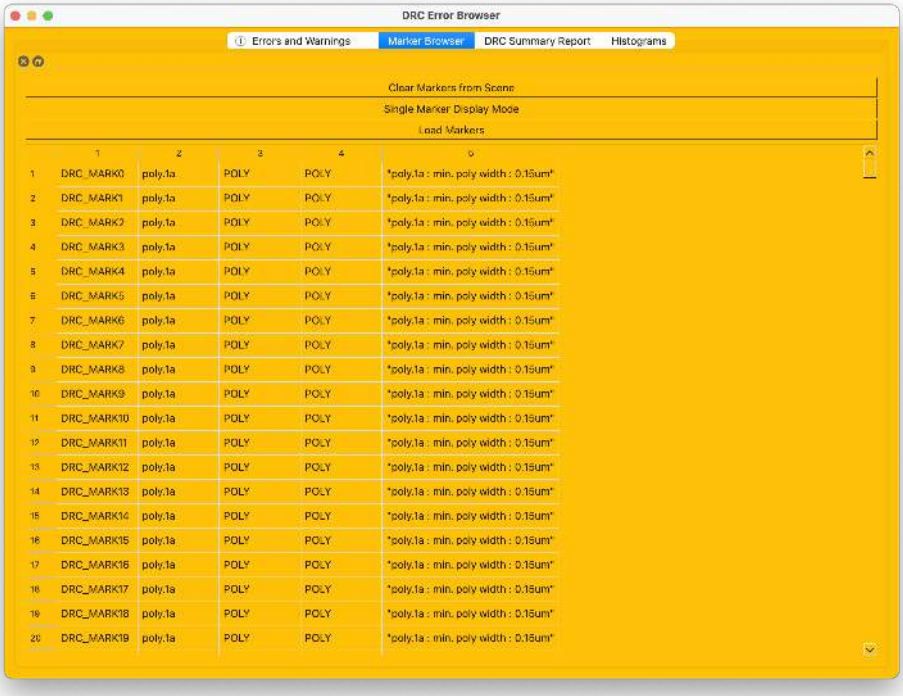


Figure 106: gcd1_DRC002_mrk_rpt.png

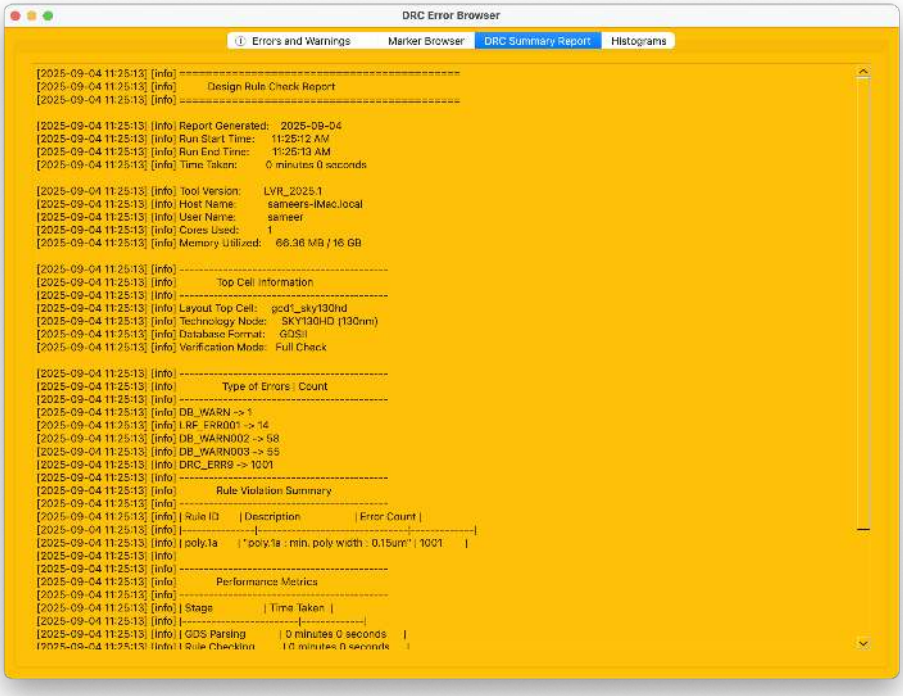


Figure 107: gcd1_DRC002_sum_rpt.png

`poly.1a` shows the total error count (in this run, on the order of 10^3), confirming widespread narrow POLY shapes across the design window.

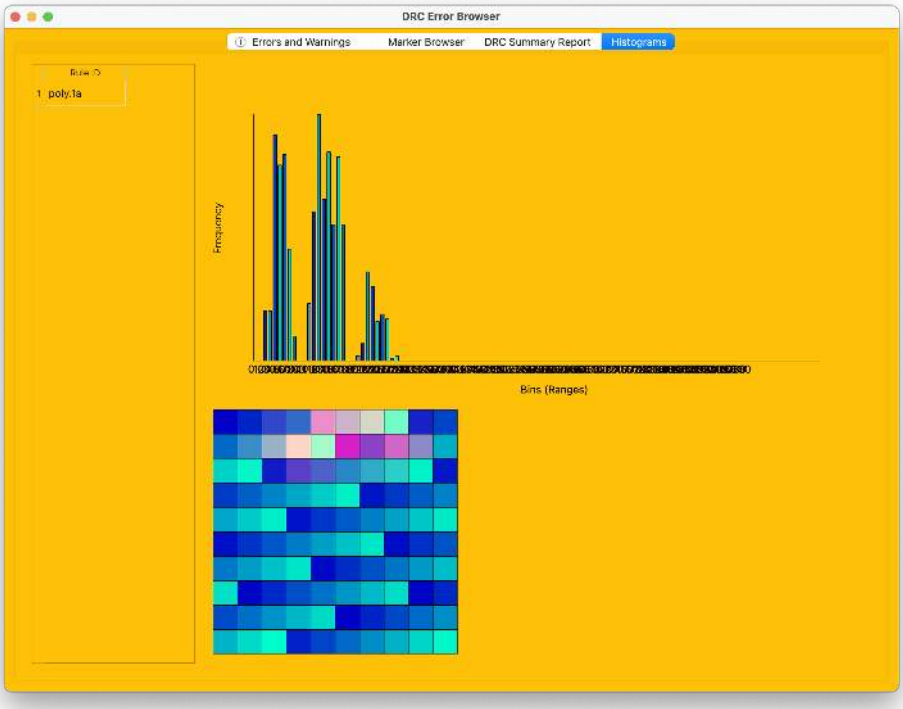


Figure 108: gcd1_DRC002_histo_rpt.png

Finally, Figure 108 shows the histogram and heat-map views. The bar chart visualizes the distribution of marker counts across spatial bins, while the heat-map quickly spots dense clusters of violations—useful to focus fixes (e.g., a repeated pattern cell producing many narrow poly lines).

2.24 Test Case: `gcd1_DRC003` (DRC)

What this test exercises. Test **gcd1_DRC003** belongs to the DRC suite. It demonstrates a Boolean-`AND`-style rule on POLY versus diffusion/tap features:

“poly must not overlap tap”. The accompanying testcase intentionally creates overlaps so the rule fires across many sites.

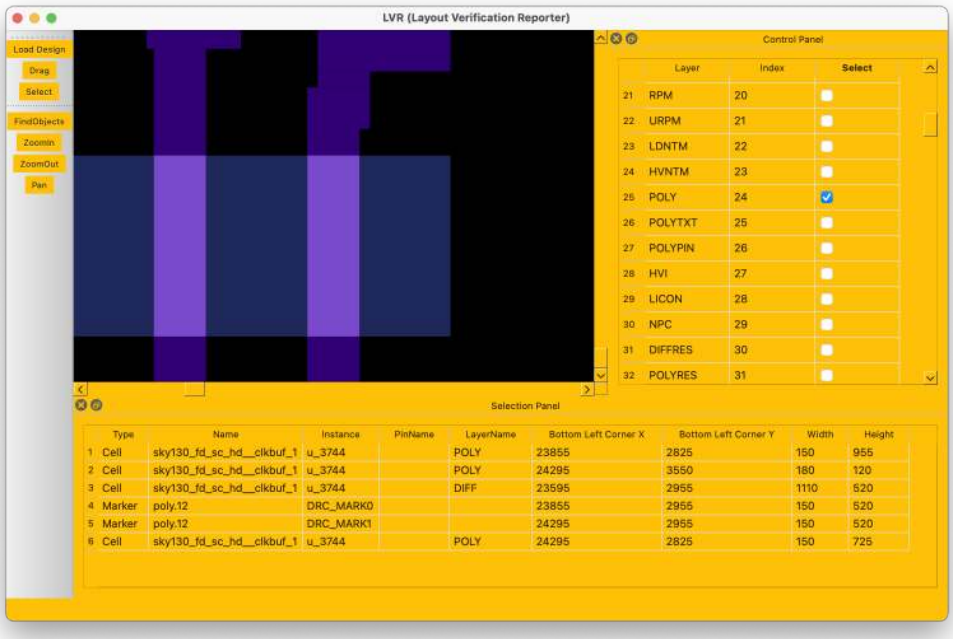


Figure 109: Use case layout view for `gcd1_DRC003`: regions where POLY intersects underlying diffusion/tap are highlighted; these overlaps are expected to violate the “poly must not overlap tap” rule.

Purpose and expected outcome. The goal is to validate that the DRC engine correctly detects all instances where a POLY shape overlaps a diffusion/tap region (Boolean `AND` of the two layers), reports them uniformly, and aggregates them in the summary, marker, and histogram reports. A successful run should: (i) complete without infrastructure errors, (ii) report nonzero violations for rule `poly.12` (caption: “poly.12 : poly must not overlap tap”), and (iii) populate the marker browser with per-site markers while the summary tallies the total count.

Injected rule settings. Figure 110 shows the relevant lines from the LRF rule file used for this testcase. The rule is expressed using a Boolean `AND` operator so that any geometric overlap between the POLY layer and a target layer produces a violation. The snippet also documents that the check is intentionally set up as a *dummy* rule to verify Boolean logic end-to-end.

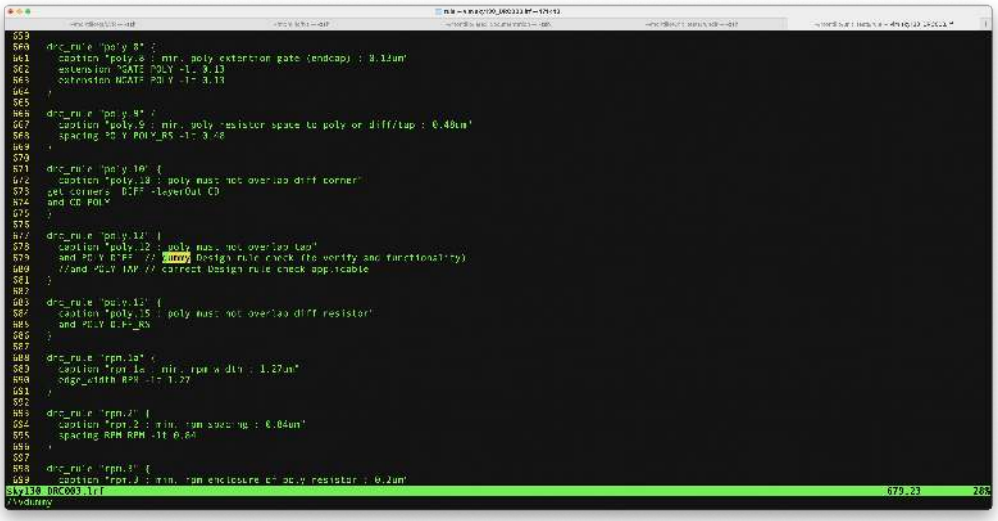


Figure 110: Error insertion / rule settings for `gcd1_DRC003`. The rule `poly.12` ("poly must not overlap tap") is implemented with a Boolean `AND` between `POLY` and a target layer, exercising the DRC Boolean engine.

Command file (verbatim). The following command file drives the run exactly as used for this testcase.

```
# initiatize tool DB
##############################
init_tool_db DRC


# set techonlogy
##############################
```

```
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130_DRC003.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db DRC
no_compare 1

##############################
# DRC settings
##############################
lvs_set_datc_flow 1

# call lvs
##############################
drc gcd1_sky130hd

##############################
# set and generate report file
##############################
```

```
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

About the settings (how LVR is configured).

- `init_tool_db DRC` selects the DRC engine and creates a DRC project database.
- `set_technode sky130` binds rule semantics and layer map to the SKY130 (130 nm) process.
- `set_top gcd1_sky130hd` names the top layout cell; reports and GUI will key off this name.
- `set_num_cores 1` confines the run to a single CPU core for reproducibility; increase for speed.
- `set_die_area 0 0 106060 106060` fixes the design window; useful for tiled/regioned runs.
- `read_lrf ../rule/sky130_DRC003.lrf` loads the rule file that contains the Boolean-`AND` `poly.12` check shown in Fig. 110.
- `read_gds ../gds/gcd1_sky130hd.gds` imports the testcase layout.
- `load_db DRC` finalizes database build for the DRC engine.
- `no_compare 1` disables LVS-style netlist comparison; only pure DRC is executed.

- `lvs_set_datc_flow 1` enables infrastructure options shared with other flows (harmless for DRC).
- `drc gcd1_sky130hd` launches the rule evaluation on the top cell.
- `load_gui` opens the GUI so you can browse markers, summaries, and histograms.

How to run. Place the commands above into a file (e.g., `gcd1_DRC003.cmd`) and invoke:

%LVR <gcd1_DRC003.cmd>

When the run finishes, open the GUI (if not auto-opened) and navigate the tabs to review results.

Describing Fig. 111. This pane shows LRF parse notes (if any) and the categorized DRC error generated by `poly.12`. Seeing this message verifies the rule is loaded and firing on overlapped polygons.

Describing Fig. 112. The marker grid enumerates every POLY overlap site. Use the GUI layer toggles to visualize the offending shapes at each coordinate to confirm the Boolean `AND` geometry.

Describing Fig. 113. This summary confirms run integrity (tool version, node, top cell) and shows the consolidated count for rule `poly.12`, matching the number of markers observed.

Describing Fig. 114. The bar chart shows how violations spread across bin ranges, while the heat-map highlights regions with higher density of POLY $\wedge$ TAP (or DIFF) overlaps, which should correlate with the patterned stimulus in Fig. 109.

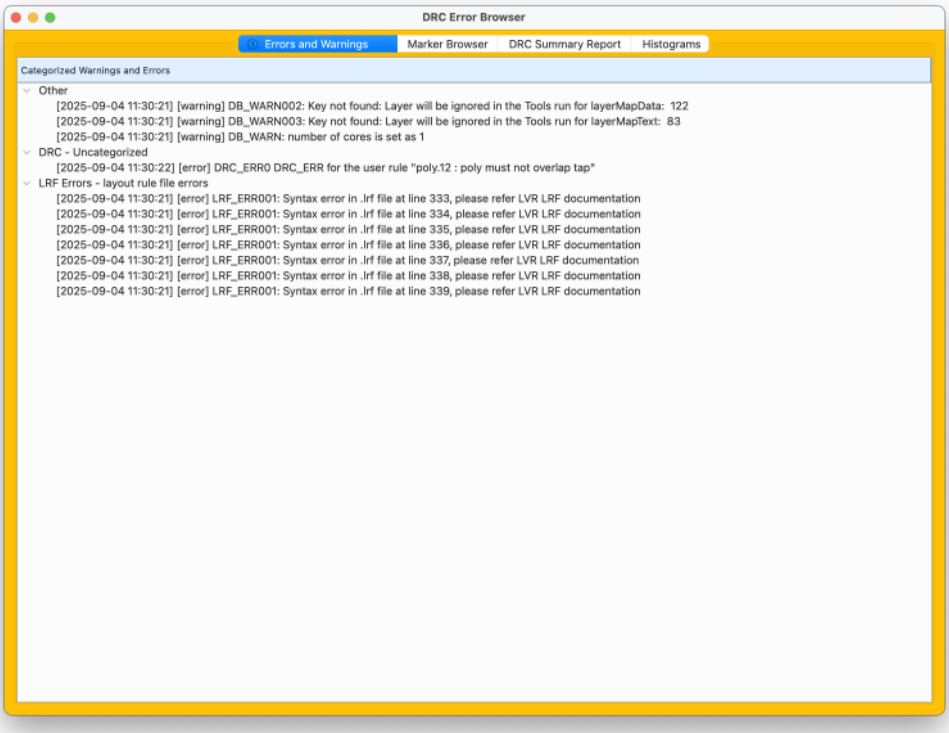
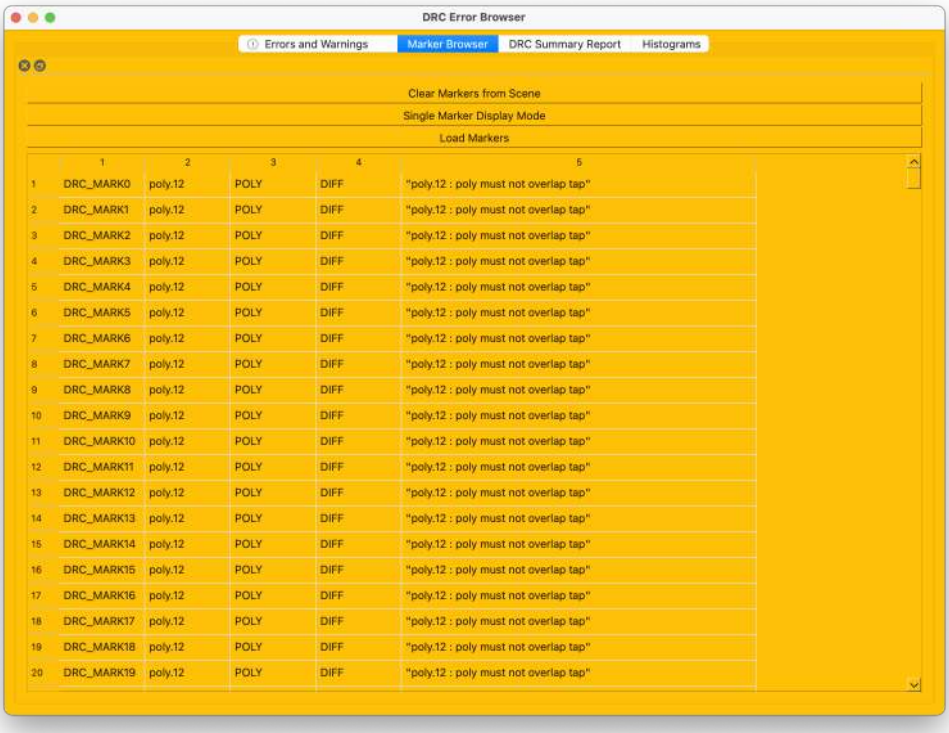


Figure 111: `gcd1_DRC003` — Errors & Warnings view. The user DRC entry lists the rule text (“poly.12 : poly must not overlap tap”), confirming the Boolean-`AND` violation is detected.

2.25 Test Case: `gcd1_DRC004` (DRC)

Intent and purpose. This testcase demonstrates an OR-style DRC rule on SKY130 where polygons on POLY are checked against diffusion resistor geometries. The rule in the injected LRF states, in effect, that *POLY must not overlap a diffusion resistor (DIFF_RS)*; for demonstration and coverage, an additional OR term is temporarily enabled so that overlaps of *POLY with generic DIFF* also trigger errors. This highlights how the LVR DRC engine evaluates boolean predicates and how violations are summarized, browsed as markers, and visualized as histograms.



	1	2	3	4	5
1	DRC_MARK0	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
2	DRC_MARK1	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
3	DRC_MARK2	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
4	DRC_MARK3	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
5	DRC_MARK4	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
6	DRC_MARK5	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
7	DRC_MARK6	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
8	DRC_MARK7	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
9	DRC_MARK8	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
10	DRC_MARK9	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
11	DRC_MARK10	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
12	DRC_MARK11	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
13	DRC_MARK12	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
14	DRC_MARK13	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
15	DRC_MARK14	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
16	DRC_MARK15	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
17	DRC_MARK16	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
18	DRC_MARK17	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
19	DRC_MARK18	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"
20	DRC_MARK19	poly.12	POLY	DIFF	"poly.12 : poly must not overlap tap"

Figure 112: `gcd1_DRC003` — Marker Browser. Each row is a violation instance of `poly.12` with the involved layers (POLY vs DIFF/TAP) and a brief description.

Figure 115 shows the small layout window used in this scenario. Intentional overlaps between POLY fingers and diffusion/diff-resistor shapes ensure the rule fires at scale, producing enough hits to exercise reporting, browsing and visualization paths.

Rule-file snippet (error injection / demonstration). The DRC rule file for this testcase contains an explicit "OR" construction to force additional, easy-to-see hits. Figure 116 captures the relevant portion of the LRF with comments indicating that the second branch is a *dummy* check used purely to demonstrate function and increase violation count.

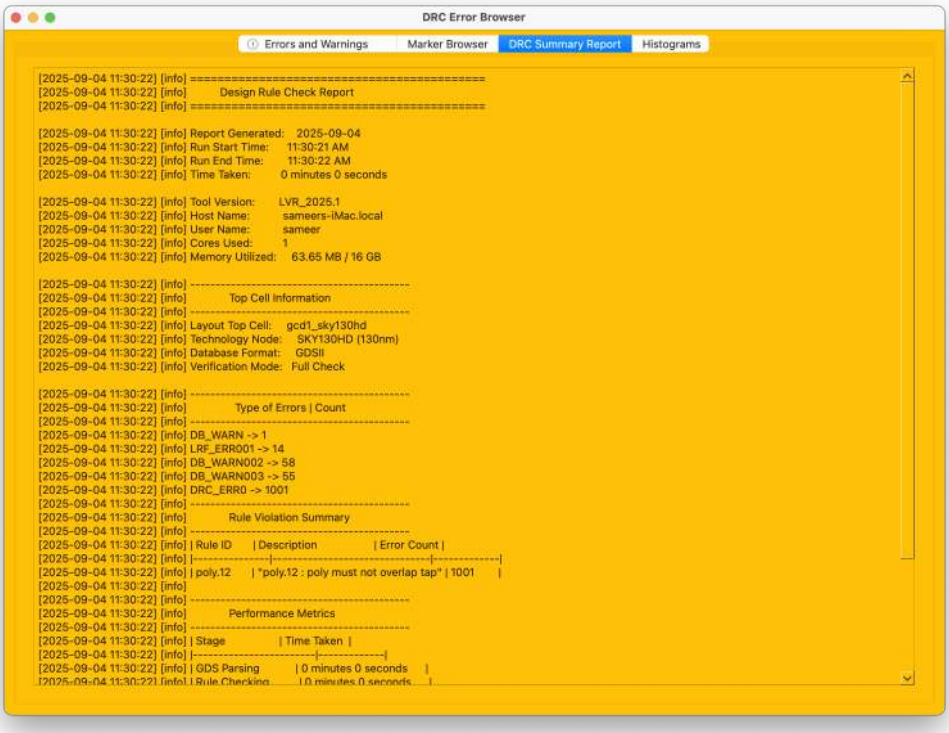


Figure 113: `gcd1_DRC003` — DRC Summary Report. The *Rule Violation Summary* table aggregates the total error count for `poly.12`, along with run metadata.

Command file. The testcase is executed through an LVR command script that initializes a DRC database, loads SKY130 rules and design GDS, and runs a DRC pass. A verbatim copy follows.

```
# initiatize tool DB
##############################
init_tool_db DRC

# set techonlogy
##############################
```

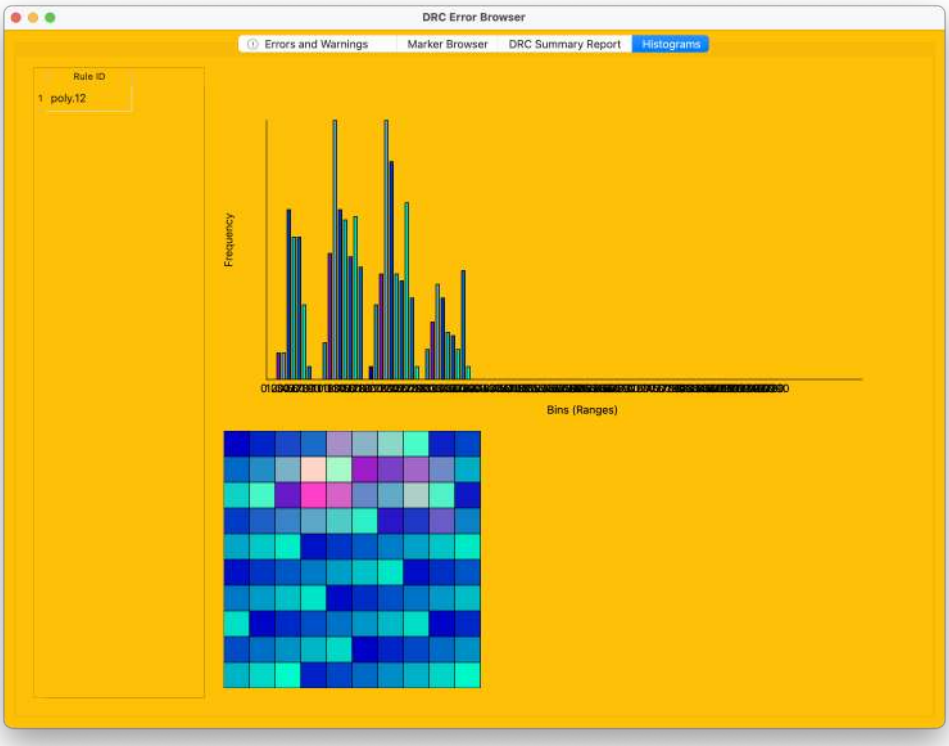


Figure 114: `gcd1_DRC003` — Histograms. Distribution and heat-map of the `poly.12` violations across bins; useful to see spatial clustering of overlaps.

```
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
```

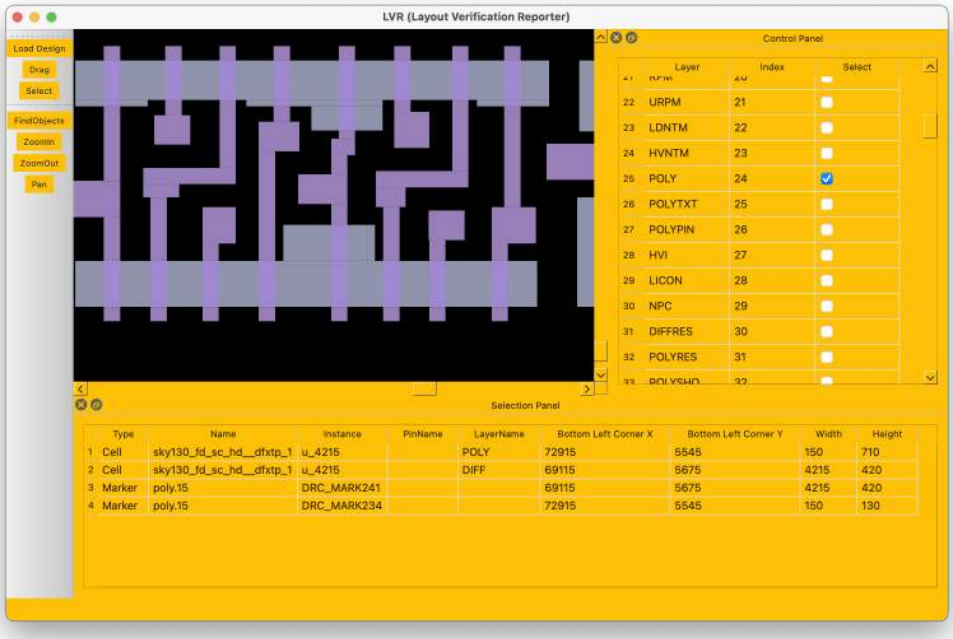


Figure 115: Use case view for `gcd1_DRC004`: intentional POLY vs DIFF/DIFF_RS geometry to exercise the OR rule.

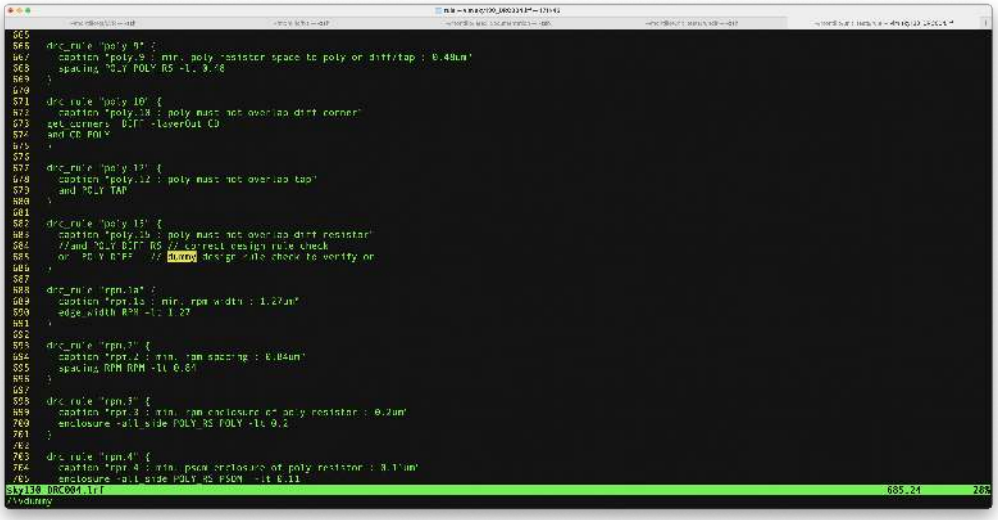


Figure 116: Injected LRF settings for `gcd1_DRC004` (“OR” demonstration). The primary check is “poly must not overlap diff resistor”; an additional OR with `POLY & DIFF` is enabled to inflate hits for documentation and visualization.

```
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130_DRC004.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db DRC
no_compare 1

##############################
# DRC settings
##############################
lvs_set_datc_flow 1

# call lvs
##############################
```

```
drc gcd1_sky130hd

############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
load_gui
```

What the settings do and why they matter.

- `init_tool_db DRC` selects the DRC pipeline and data model.
- `set_technode sky130` binds technology semantics and default grids to the SKY130 process.
- `set_top gcd1_sky130hd` defines the top-level cell used for rule evaluation and report headers.
- `set_num_cores 1` keeps the run single-threaded (repeatable timing/ordering); `set_die_area` scopes the scene.
- `read_lrf ../rule/sky130_DRC004.lrf` loads the dedicated rule deck containing the OR-based check for this demo.
- `read_gds ../gds/gcd1_sky130hd.gds` ingests the layout; `load_db DRC` finalizes the DRC view.
- `no_compare 1` disables netlist/schema comparisons (not needed for pure DRC).

- `lvs_set_datc_flow 1` enables a shared parsing/runtime path used across LVR apps; harmless here and keeps runs uniform.
- `drc gcd1_sky130hd` executes the rule deck on the top cell.
- `load_gui` launches the interactive GUI to review markers, summaries, and histograms.

How to run. From an LVR shell, invoke:

```
% LVR gcd1_DRC004.cmd
```

This runs the script end-to-end, generates the reports, and opens the GUI (unless suppressed). Reports and views referenced below are produced by this exact invocation.

Results overview and screenshots. Figure 117 shows the Errors & Warnings pane summarizing categorized issues. Here the dominant item is the OR-rule violation on POLY vs DIFF/DIFF_RS, exactly as constructed in the rule file.

Figure 118 lists the marker table populated by the run. Each row corresponds to one spatial hit with rule ID `poly.15` (caption: "poly must not overlap diff resistor"), layer context, and coordinates, making it easy to navigate to specific offenders.

A top-level tally appears in Figure 119, where the violation summary aggregates counts by rule ID. You should see `poly.15` dominate, corroborating the expected behavior for this demonstration rule.

Finally, Figure 120 presents the histogram view, which is useful for spotting spatial clustering and distribution trends of the violations across bins/ranges.

Key takeaways. This testcase validates how an OR condition in the rule deck expands the violation set and how LVR presents those results. It is a compact template for authoring boolean-style checks in SKY130, reviewing

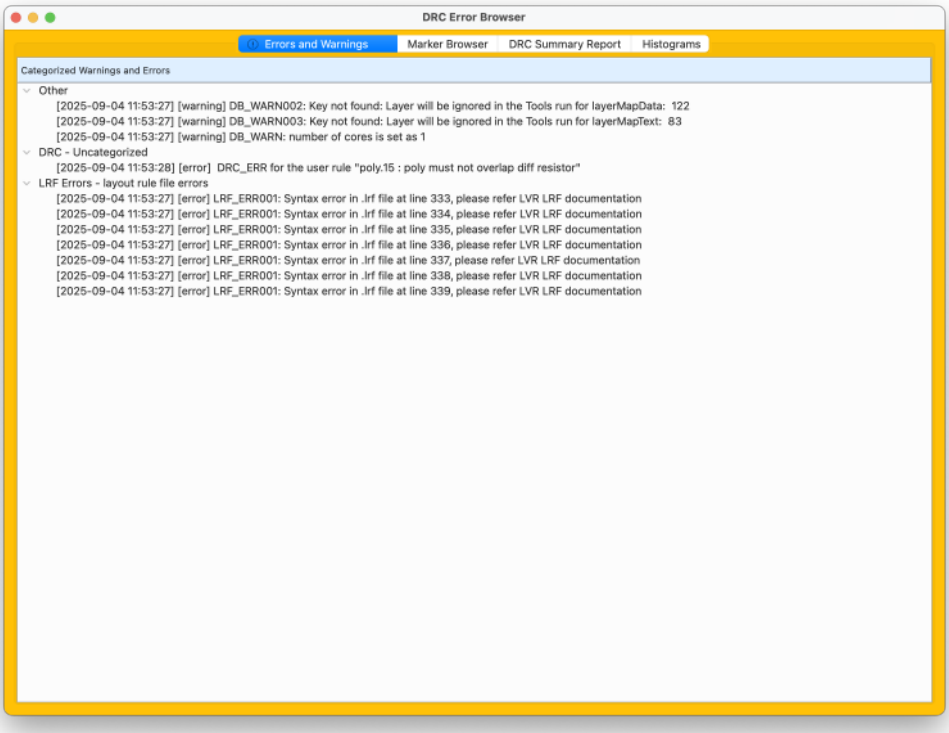


Figure 117: Errors view for `gcd1_DRC004`: entries reflect the OR-based overlap rule.

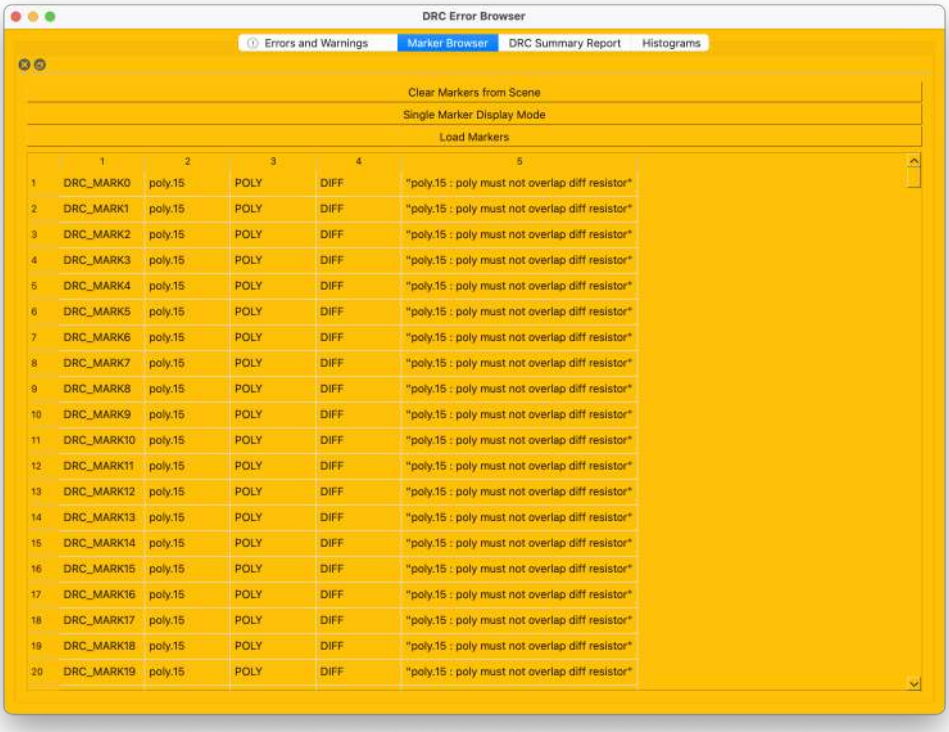


Figure 118: Marker Browser for `gcd1_DRC004`: individual violation instances with rule name, layers, and locations.

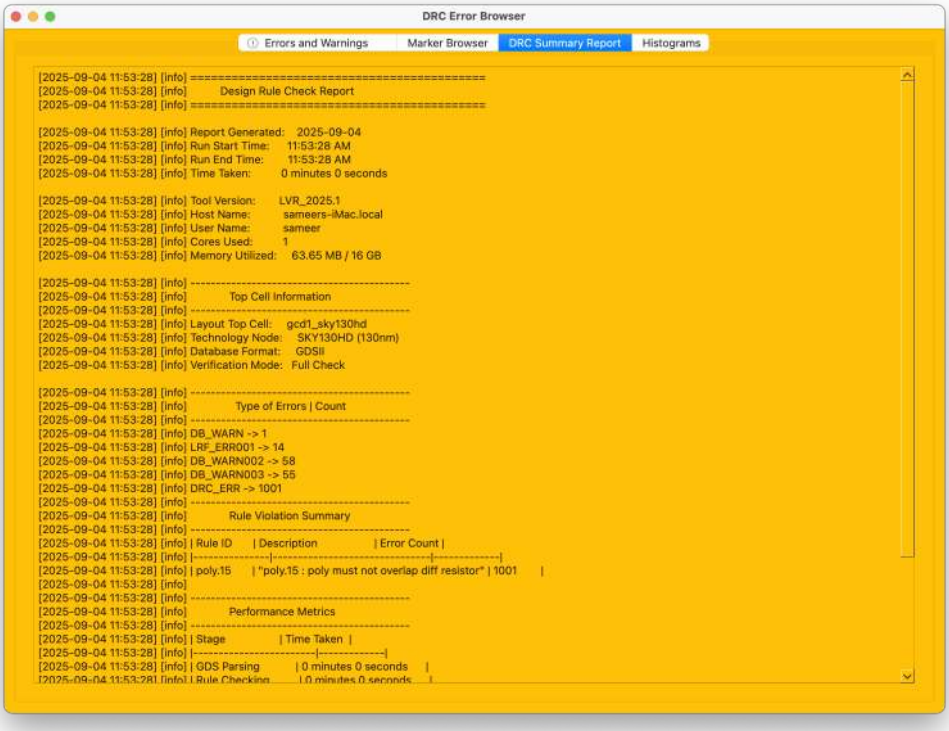


Figure 119: DRC Summary Report for `gcd1_DRC004`: aggregated counts by rule; `poly.15` is the key item.

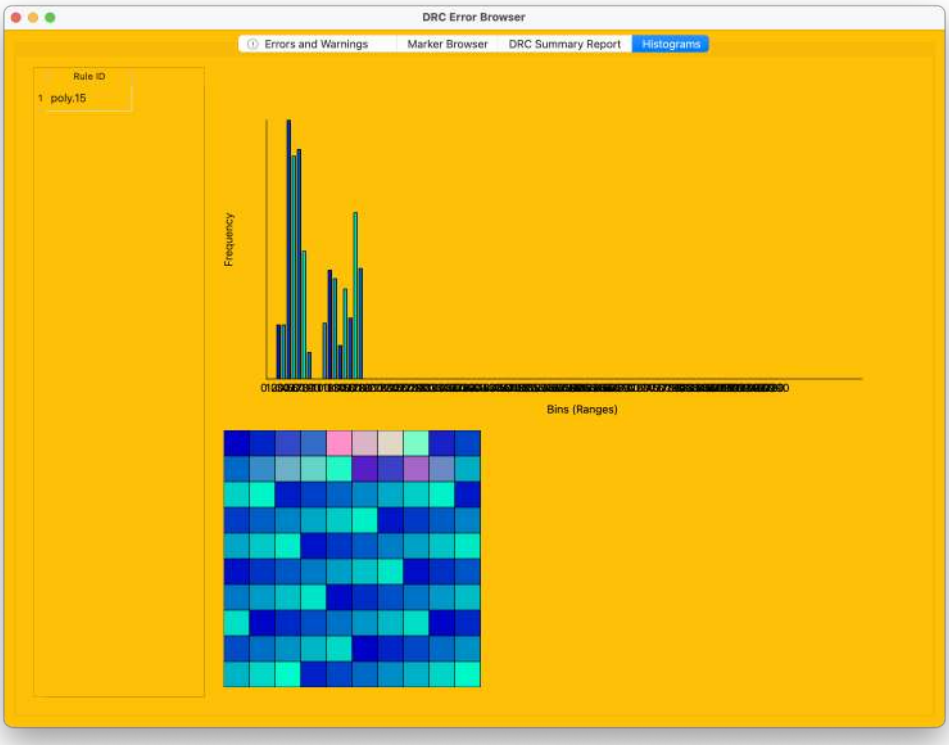


Figure 120: Histogram view for `gcd1_DRC004`: frequency and heatmap visualization of the OR-rule violations.

hits in the marker browser, confirming totals in the summary report, and visualizing distribution via histograms. Replace the dummy OR branch with your production predicate to switch from demo to signoff behavior.

2.26 Test Case: `gcd1_DRC005` (DRC)

Context. From its name, `gcd1_DRC005` is a *Design Rule Check (DRC)* testcase. It exercises a NOT-style geometric predicate inside the Sky130 rule deck to flag transistors whose LI/DIFF/Poly topology violates the minimum source/drain length requirement.

What this testcase demonstrates. This testcase injects an intentional violation of the poly rule `poly.7 : min. source/drain length : 0.25` using a NOT construct in the LRF (rule) file. The design places standard cells so that some polysilicon gates create short source/drain segments. The rule-layer logic in the deck uses negative (NOT) conditions around DIFF and POLY features at transistor `source` and `drain` shapes to detect undersized segments, then computes the edge width against a threshold. As a result, running DRC on this layout reports a large set of markers on the poly layer where the minimum source/drain length is not met.

Rule file snippet and error insertion. Figure 122 shows the relevant portion of the Sky130 LRF used only for this testcase. The deck documents it explicitly as a *dummy* (deliberate) error injection to validate the NOT logic: it temporarily changes the evaluation by adding forced `edge_width` constraints on the `source` and `drain` shapes and uses negative selections (`not DIFF POLY -layerOut source`, etc.) so that short segments fail the check.

How to run and what to expect. Running the supplied command file executes a full DRC pass with the Sky130 deck variant `sky130_DRC005.lrf`. The summary report will list one violated rule, `poly.7`, with a high count of

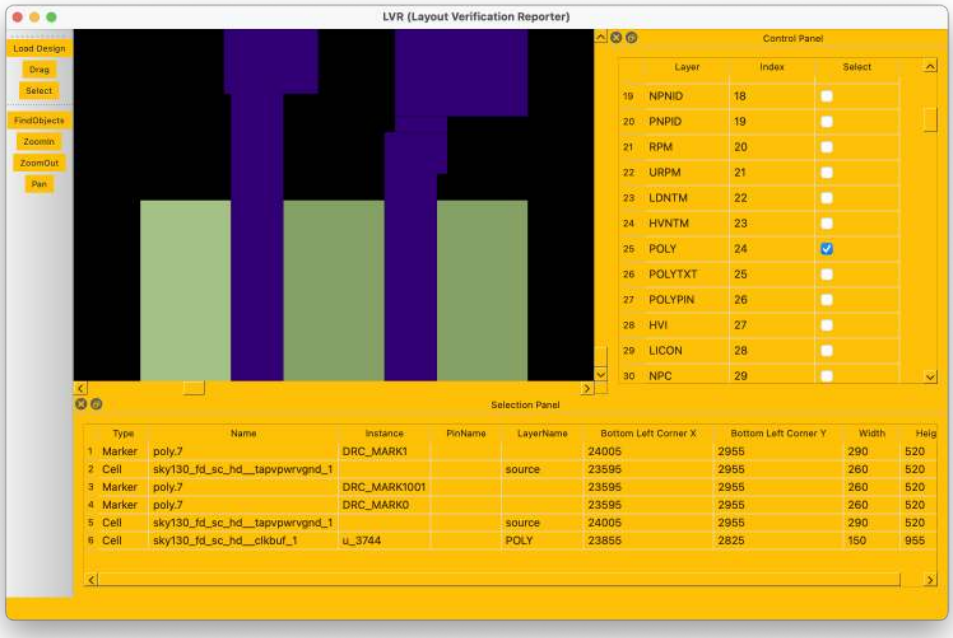


Figure 121: **Use case view** for `gcd1_DRC005`: polysilicon (purple) crossing diffusion with taps beneath; the highlighted regions are where the NOT-based rule will trigger.

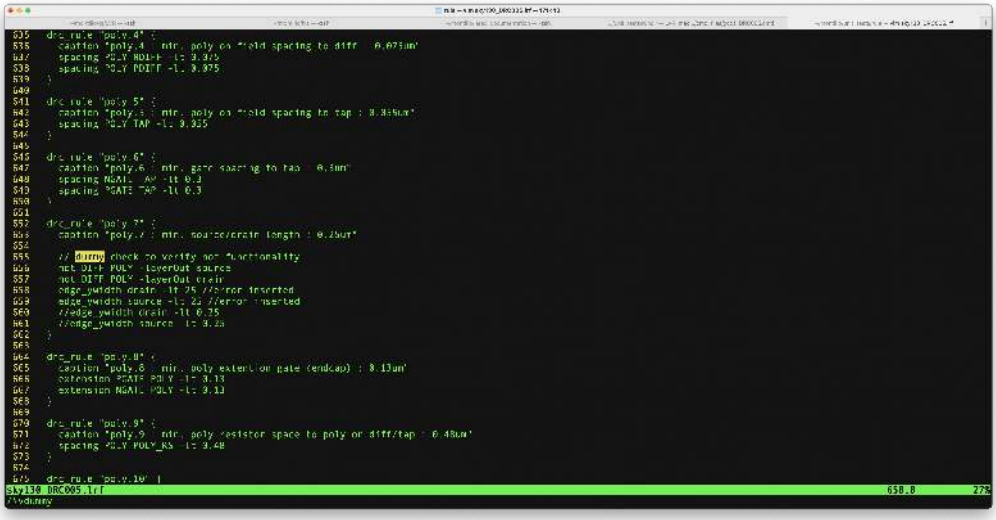


Figure 122: Rule-file excerpt for `poly.7` in `sky130_DRC005.lrf` showing the NOT-based selection and injected thresholds used to provoke violations for this testcase.

errors; the marker browser will contain per-violation entries, and histograms will bin marker counts by coordinate ranges, useful for visualizing clustering (Figures 123, 124, 125, and 126).

Command file (`gcd1_DRC005.cmd`). A verbatim copy is provided below.

```
# initiatize tool DB
##############################
init_tool_db DRC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
```

```
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130_DRC005.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db DRC
no_compare 1

##############################
# DRC settings
##############################
lvs_set_datc_flow 1

# call lvs
##############################
drc gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

##############################
```

```
# load gui
############################
load_gui
```

What the settings do (one-page overview).

- `init_tool_db DRC` selects the DRC application and initializes an empty database with rule-checking kernels loaded.
- `set_technode sky130` binds technology defaults (layer aliases, derived layers, units) for SkyWater 130 nm.
- `set_top gcd1_sky130hd` sets the top-level cell name that subsequent commands reference (both for loading and for checks).
- `set_num_cores 1` pins execution to one core (deterministic runs, easier log comparison). For throughput, you can raise this.
- `set_die_area 0 0 106060 106060` provides an explicit design window (in database units) so tiling/regioning and histogram binning are consistent across runs.
- `read_lrf ../rule/sky130_DRC005.lrf` loads the deck variant that contains the NOT-based `poly.7` rule and its deliberate threshold tweaks used by this testcase.
- `read_gds ../gds/gcd1_sky130hd.gds` brings in the layout database in GDSII.
- `load_db DRC` finalizes the geometry store for rule evaluation; `no_compare 1` disables LVS/DB comparison phases so only DRC runs.
- `lvs_set_datc_flow 1` keeps common data/control-path scaffolding enabled; it does not change the active app (still DRC) but preserves uniform logging.

- `drc gcd1_sky130hd` executes the full DRC rule deck on the top cell.
- `load_gui` opens the GUI (Error Browser and Layout Viewer) to browse markers, summaries, and histograms.

How to run. From a shell where `LVR` is on your PATH, invoke:

$$\% \texttt{LVR} <\texttt{gcd1\_DRC005.cmd}>$$

This runs the sequence above, produces a DRC Summary Report, and populates the GUI browser tabs. Logs and report artifacts appear in the working directory unless redirected by the (commented) report settings.

2.27 Test Case: `gcd1_DRC006` (DRC)

Context. From the test name `gcd1_DRC006`, the verification context is **DRC** (Design Rule Check), i.e., $context = \mathrm{DRC}$.

Objective and use-case. This testcase demonstrates how to author and exercise a DRC that relies on the `get_corners` geometric operator. In SKY130 CMOS, one frequent pattern to guard against is polysilicon overlapping or intruding into diffusion *corners*. The rule under test flags POLY that touches any corner extracted from DIFF, and serves primarily as a functional check of the rule language (LRF) operator pipeline and marker rendering in LVR.

Figure 127 shows the small stimulus layout excerpt used in this scenario. The scene contains diffusion blocks with varied corner topologies and overlying POLY features crafted to exercise positive and negative cases for the corner detector.

Rule implementation snapshot (error injection / configuration). Figure 128 is an excerpt from the LRF where `get_corners` is used to form the DIFF corner set and then intersected with POLY to realize the check "*poly must not overlap*

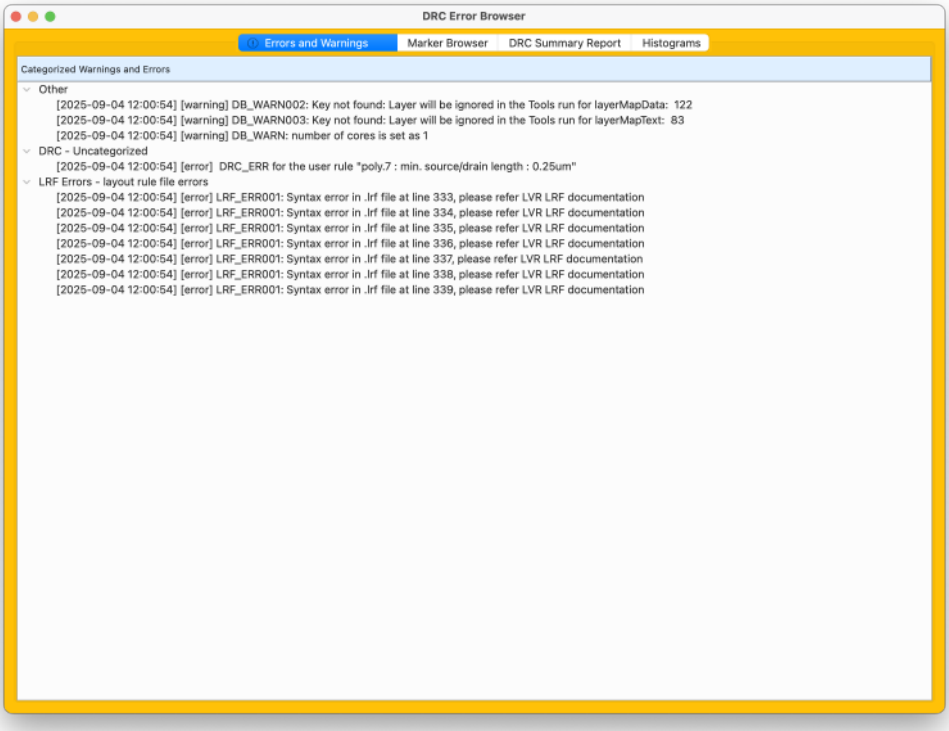


Figure 123: Errors & Warnings tab for `gcd1_DRC005`. The category *DRC — Uncategorized* lists violations for rule `poly.7 : min. source/drain length : 0.25`; LRF parse warnings shown below come from intentionally edited rule lines used by this testcase.

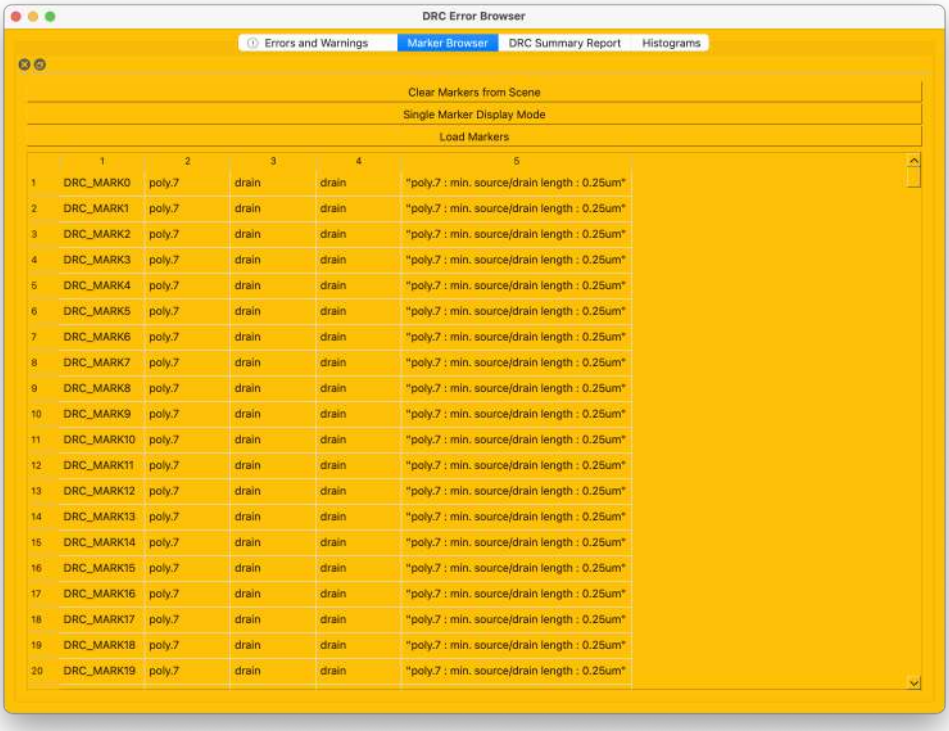


Figure 124: Marker Browser for `gcd1_DRC005`. Each row is a DRC marker for `poly.7`; columns show rule ID, involved layers (`POLY`, `DIFF`), and a human-readable description (“min. source/drain length : 0.25 ”).

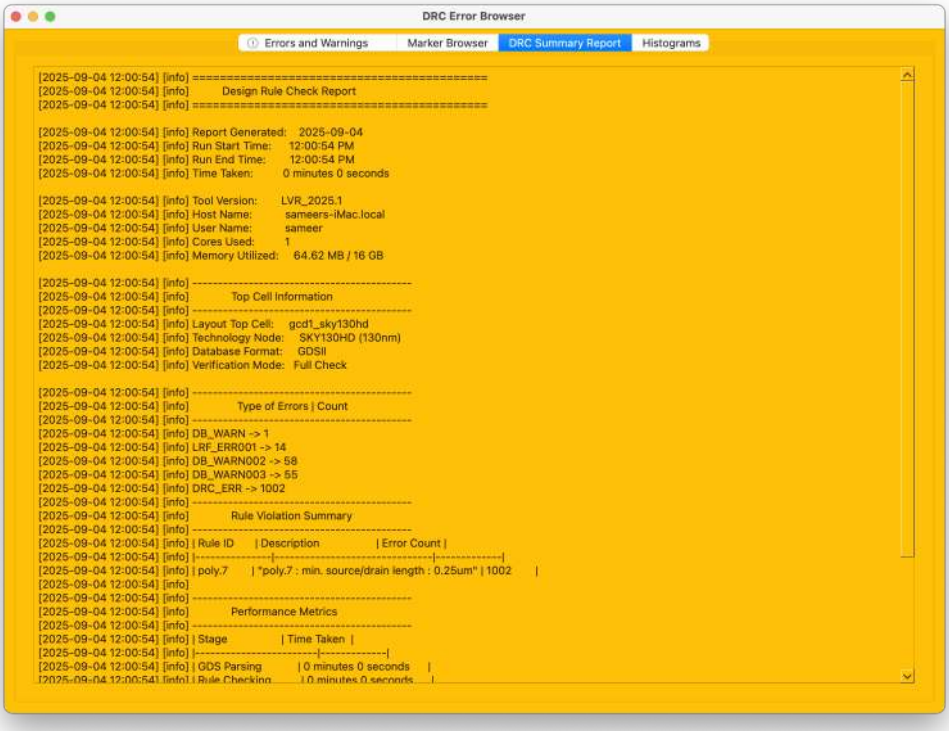


Figure 125: DRC Summary Report for `gcd1_DRC005`. It lists the top cell, technology, verification mode, and the rule-violation summary showing a high error count for `poly.7`, as expected from the injected NOT-based check.

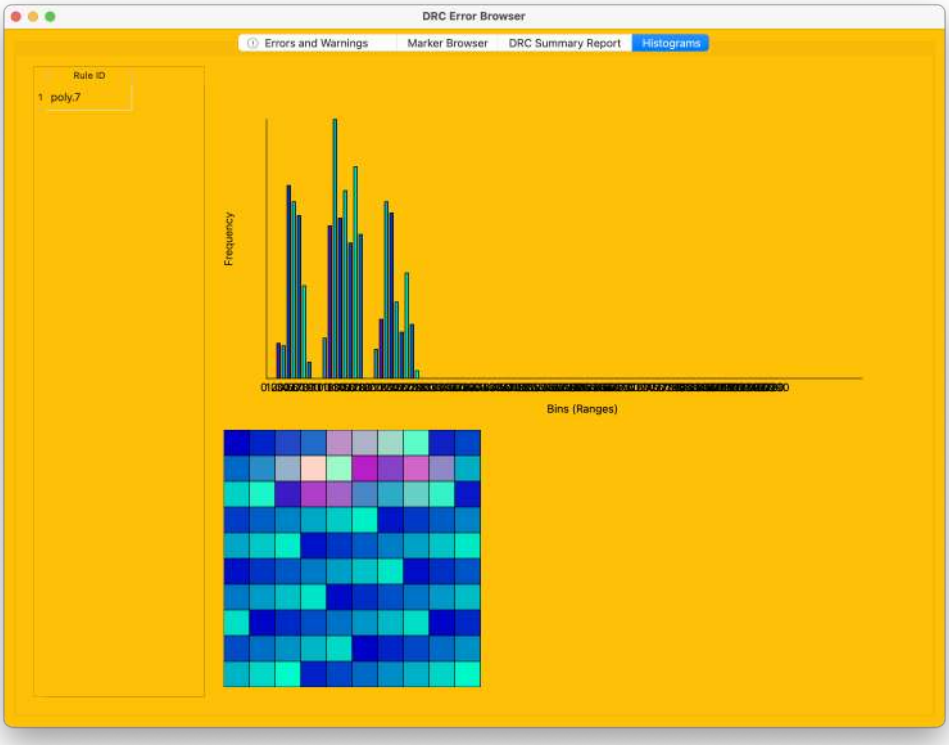


Figure 126: Histograms for `gcd1_DRC005`. The upper chart bins violation counts across the die (by range), revealing clustering where short source/drain segments are repeated; the heatmap below offers a spatial density cue for the same markers.

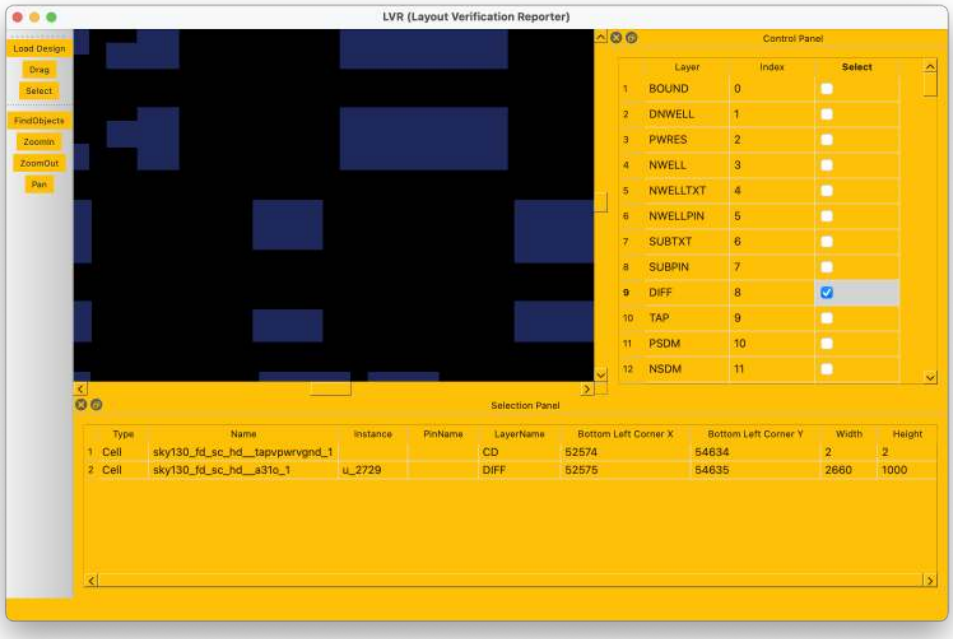


Figure 127: gcd1_DRC006 use case layout shown in LVR.

diff corner.” This is the crux of the testcase: confirming that the `get_corners` set is built and consumed correctly within the rule.

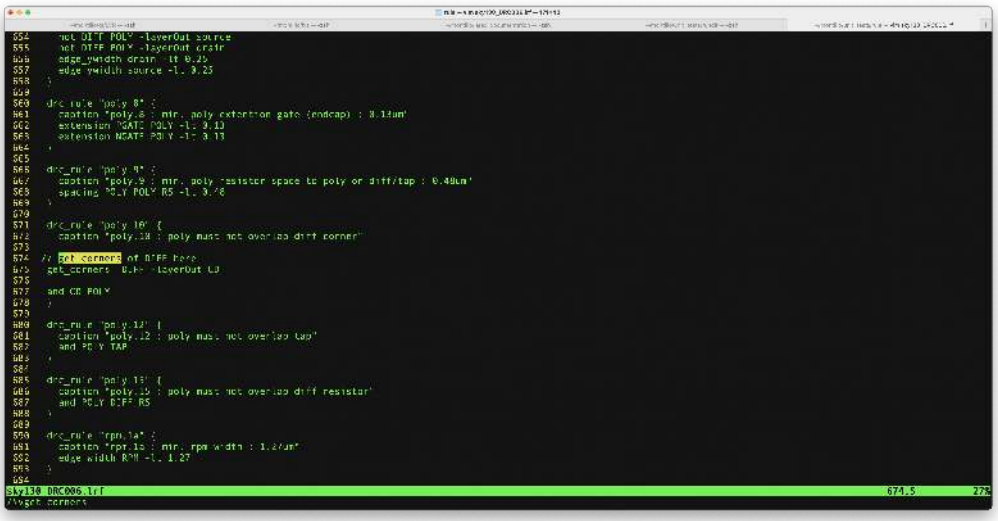


Figure 128: LRF snippet for `get_corners`: build DIFF corners and test overlap against POLY.

Command file. The following command file drives the check end-to-end in LVR.

Command file `gcd1_DRC006.cmd` **(verbatim)**

```
# initiatize tool DB
##############################
init_tool_db DRC

# set techonlogy
##############################
set_technode sky130
```

```
# set top module
##############################
set_top gcd1_sky130hd

# set number of cores
##############################
set_num_cores 1
set_die_area 0 0 106060 106060
#set_gds_flow 1
# read the rule file and gds
##############################
read_lrf ../rule/sky130_DRC006.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db DRC
no_compare 1

##############################
# DRC settings
##############################
lvs_set_datc_flow 1

# call lvs
##############################
drc gcd1_sky130hd

##############################
# set and generate report file
##############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs
```

```
############################
# load gui
############################
load_gui
```

What the settings do and why they matter

- `init_tool_db DRC` selects the DRC flow, enabling rule-driven polygon ops and marker generation.
- `set_technode sky130` binds all unit scaling and layer semantics to the SKY130 PDK assumptions used by the LRF.
- `set_top gcd1_sky130hd` names the top layout cell for context and reporting.
- `set_num_cores 1` and `set_die_area ...` keep the run deterministic and scoped to the GDS region containing the stimulus.
- `read_lrf ../rule/sky130_DRC006.lrf` ingests the rule file whose critical portion is the `get_corners`-based rule (Fig. 128).
- `read_gds ...` loads the test layout; `load_db DRC` finalizes the database for checking.
- `no_compare 1` disables schematic/layout comparisons—this is a pure DRC run.
- `lvs_set_datc_flow 1` leaves the data-conditioning path enabled so that rule operations see a clean, normalized polygonal model.
- `drc gcd1_sky130hd` executes the check and generates markers for any POLY $\cap$ `get_corners(DIFF)` overlaps.
- `load_gui` opens the interactive browsers to visualize the results.

How to run. From an LVR shell (or your run directory), invoke:

$$\% \ \mathrm{LVR} \ \langle \texttt{gcd1\_DRC006.cmd} \rangle$$

i.e., `% LVR gcd1_DRC006.cmd`. This parses the LRF, loads the GDS, runs the DRC, and pops up the GUI with Error/Marker/Summary/Histogram tabs.

Reports and screenshots

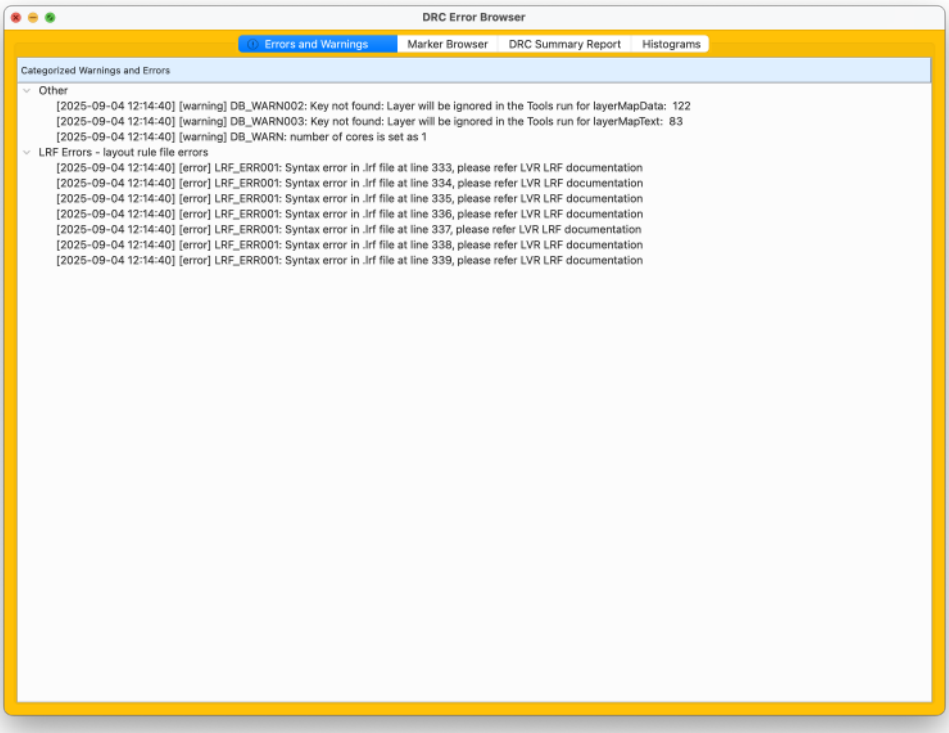


Figure 129: gcd1_DRC006: Errors & Warnings tab.

About Fig. 129. The run shows parser notices for generic LRF lines (expected in this unit test environment) and, for this stimulus, no user DRC errors tied to

the `get_corners` rule—useful to confirm that correct-by-construction cases remain clean.

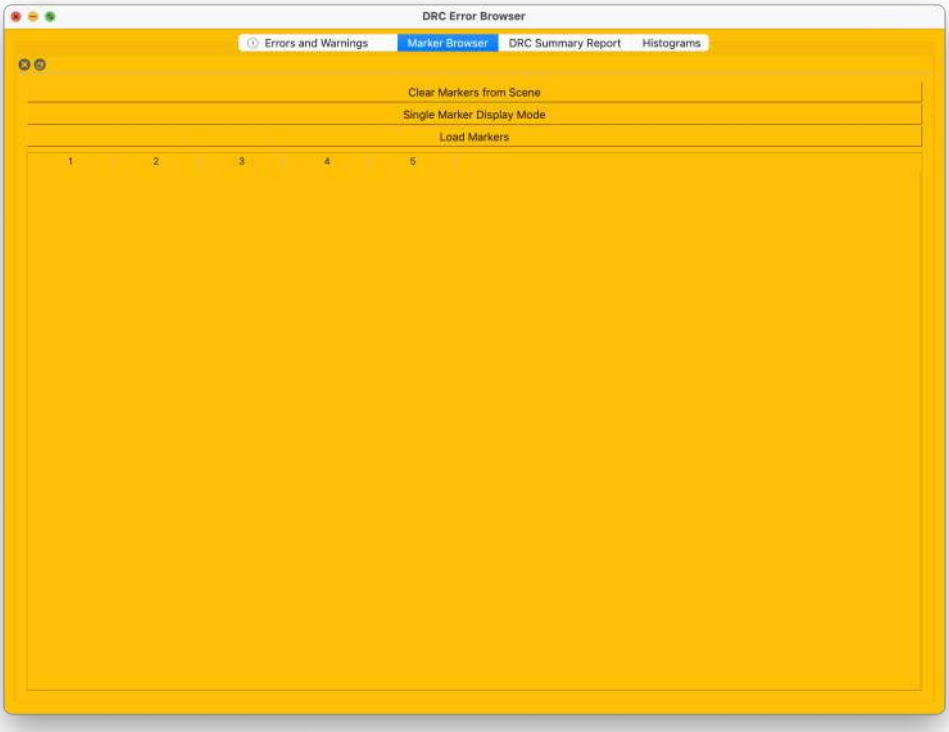


Figure 130: gcd1_DRC006: Marker Browser.

About Fig. 130. The Marker Browser is empty for this run, indicating no POLY overlaps were found at DIFF corners (i.e., the rule did not trigger). When violations exist, this table lists each marker with affected layers and human-readable captions.

About Fig. 131. The summary page enumerates global run settings and would normally include a “Rule Violation Summary.” Its absence here corroborates the clean outcome for this corner-overlap check.

About Fig. 132. With no rule hits, the histogram panel shows no distributions. In failing cases, this view is handy to understand hotspot clustering and error

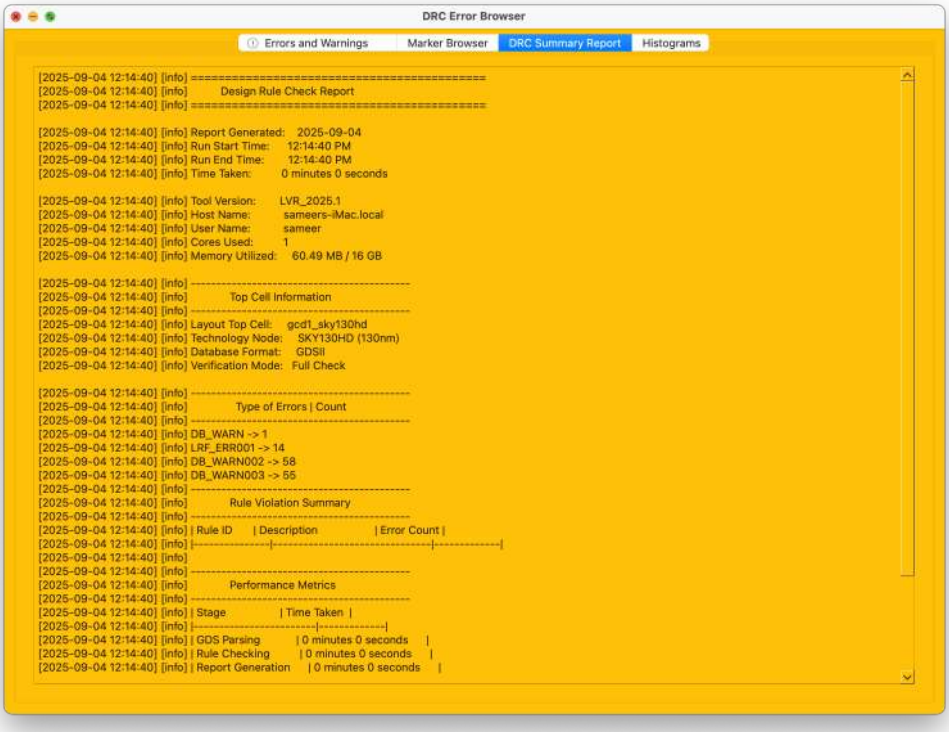


Figure 131: gcd1_DRC006: DRC Summary Report.

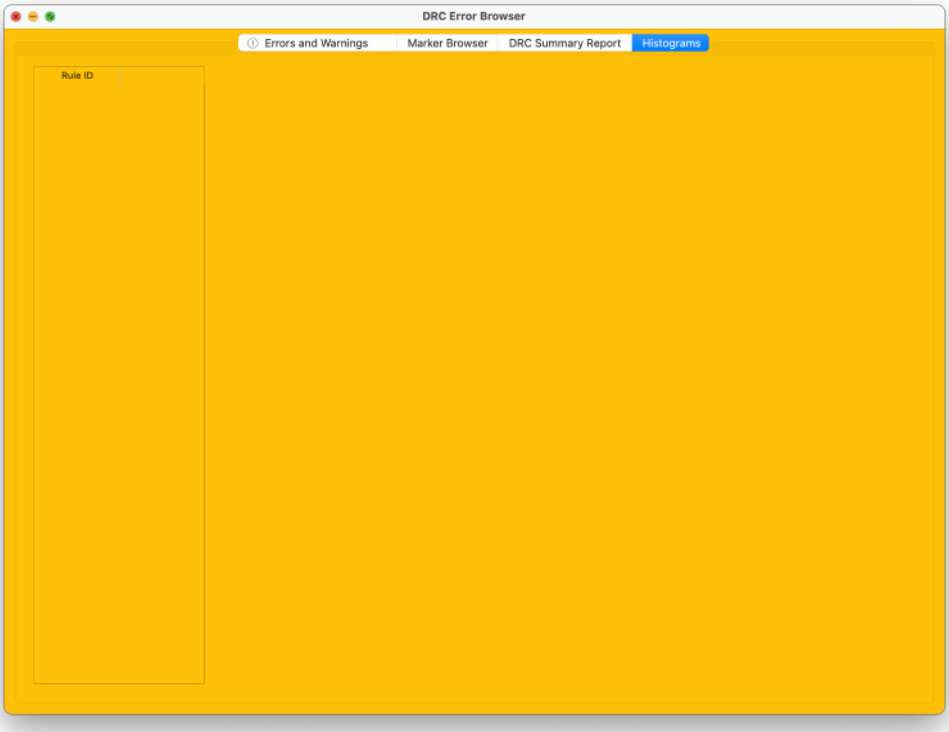


Figure 132: gcd1_DRC006: Histograms view.

spread across bins/windows.

Takeaways. This testcase is a concise template for writing and validating DRCs that depend on derived geometry (`get_corners`). Even when the intent is a "no-error" golden check, wiring it into the normal flow (rule load $\rightarrow$ run $\rightarrow$ browse) ensures that database conditioning, rule evaluation, reporting, and GUI linkage are all exercised, making it easy to extend toward failing stimuli or additional corner classes later.

2.28 Test Case: `gcd1_DRC007` (DRC)

Purpose and scenario. This testcase demonstrates a *minimum edge length* design rule on the `NWELL` layer. The goal is to show how LVR (Layout Verification Reporter) flags regions of `NWELL` whose polygon edge length is below the rule threshold (here, 0.84 μm in technology units). The check is intentionally exercised on a macro-level layout to verify rule authoring, report quality (errors, markers, histograms), and interactive debug in the LVR GUI.

Figure 133 shows the LVR viewer with `NWELL` selected (right-side *Control Panel*) and the selection table at the bottom listing a fill cell instance and the associated DRC marker. The pink bands indicate the `NWELL` geometry segmenting the die, and markers are placed where the minimum edge length rule is violated.

Rule instrumentation (error injection). To make the testcase self-checking and visually obvious, the rule deck includes an explicit edge-length constraint on `NWELL`. The relevant portion of the LRF file is summarized in Figure 134. The rule is named `dummy_nwell.1.L` with caption "*nwell.1.L : min. nwell length : 0.84um*" and uses an `edge_length` operator on `NWELL`. Database units (DBU) are used internally; the value `8400` corresponds to $0.84\,\mu\mathrm{m}$ when DBU = 1 nm.

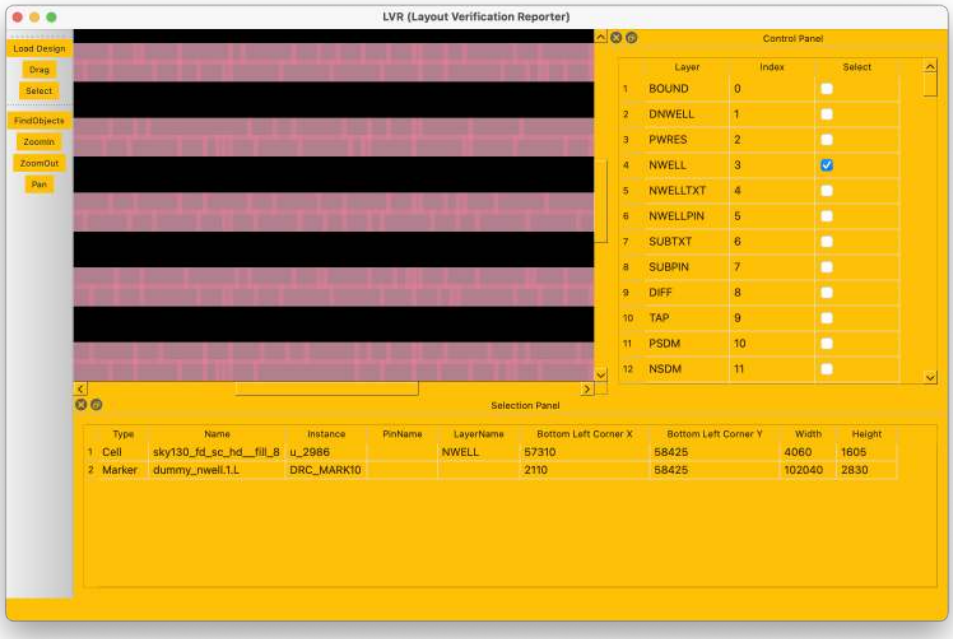


Figure 133: **Use case view** for `gcd1_DRC007`: `NWELL` edge-length violations highlighted in the LVR viewer.

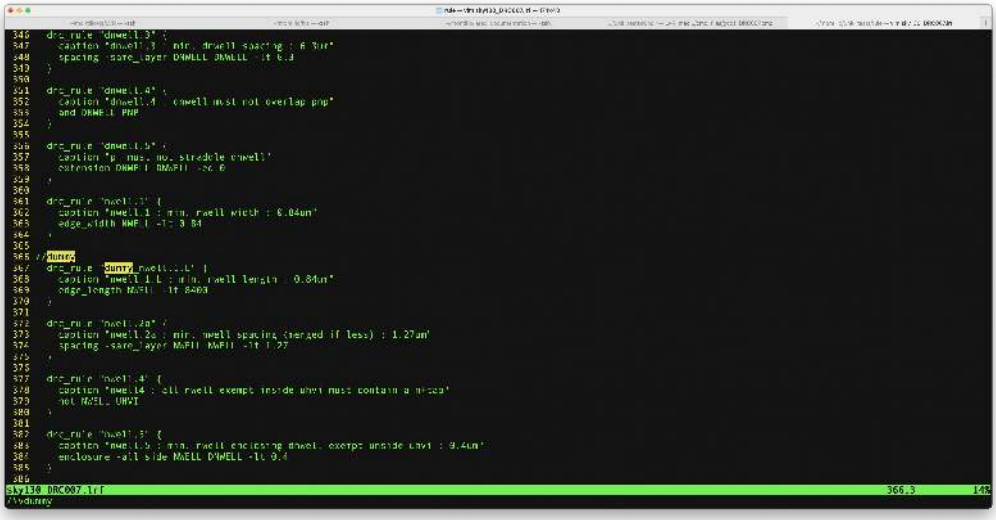


Figure 134: **Error-injection rule snippet** for `gcd1_DRC007`: the LRF uses `edge_length NWELL -lt 8400` to trigger violations below 0.84 μm.

Command file. The test is driven by the following `.cmd` file.

Command file: `gcd1_DRC007.cmd` **(verbatim)**

```
# initiatize tool DB
##############################
init_tool_db DRC

# set techonlogy
##############################
set_technode sky130

# set top module
##############################
set_top gcd1_sky130hd
```

```
# set number of cores
############################
set_num_cores 1
set_die_area 0 0 106060 106060
#set_gds_flow 1
# read the rule file and gds
############################
read_lrf ../rule/sky130_DRC007.lrf
read_gds ../gds/gcd1_sky130hd.gds
load_db DRC
no_compare 1

############################
# DRC settings
############################
lvs_set_datc_flow 1

# call lvs
############################
drc gcd1_sky130hd

############################
# set and generate report file
############################
#set_report_file_lvs lvs_summary.rpt
#report_lvs

############################
# load gui
############################
```

```
load_gui
```

What the settings do (one-page overview).

- `init_tool_db DRC` creates a fresh LVR database in DRC mode.
- `set_technode sky130` selects the SkyWater SKY130 technology; layer semantics and default units come from the tech setup.
- `set_top gcd1_sky130hd` names the layout's top cell used for rule evaluation and reporting.
- `set_num_cores 1` runs single-threaded (deterministic and lightweight for the example).
- `set_die_area 0 0 106060 106060` declares a die window (in DBU) for region-of-interest clipping and tiling.
- `read_lrf ../rule/sky130_DRC007.lrf` loads the DRC rule deck containing the `edge_length` rule on `NWELL`.
- `read_gds ../gds/gcd1_sky130hd.gds` imports the test GDS.
- `load_db DRC` finalizes DB assembly for checking; `no_compare 1` ensures no LVS/compare phase is attempted.
- `lvs_set_datc_flow 1` enables the data-conversion/tiling flow used internally by LVR for robust runs on large designs (safe to keep on for small examples).
- `drc gcd1_sky130hd` launches the DRC engine for the top cell.
- Optional report emitters (`set_report_file_lvs`/`report_lvs`) are shown but commented; the GUI will still expose the summary, markers, and histograms.
- `load_gui` opens the interactive LVR GUI for result browsing.

How to run. From your LVR shell, execute:

```
%LVR gcd1_DRC007.cmd
```

This runs the flow described above and opens the results in the GUI.

Run results. The following figures capture the principal result views produced by this testcase.

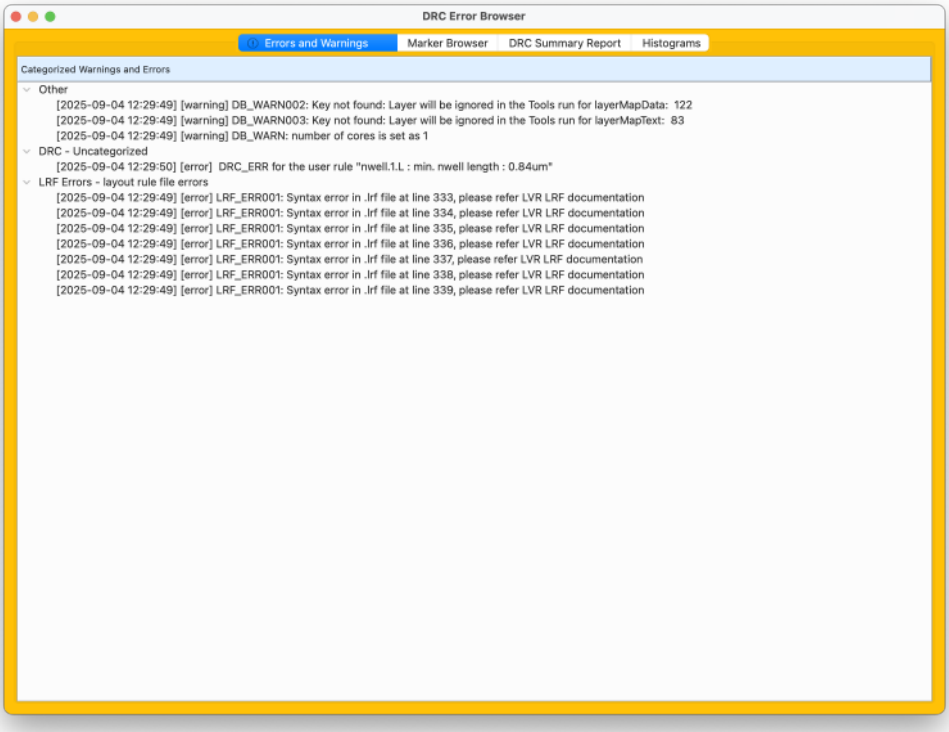


Figure 135: **Errors & Warnings view** for `gcd1_DRC007`. The pane lists categorized messages; note the DRC entry for rule “nwell.1.L : min. nwell length : 0.84um” and any LRF syntax notes (if present).

Interpreting the screenshots. Figure 135 confirms the rule firing (“min. nwell length : 0.84um”). Figure 136 enumerates all marker instances with

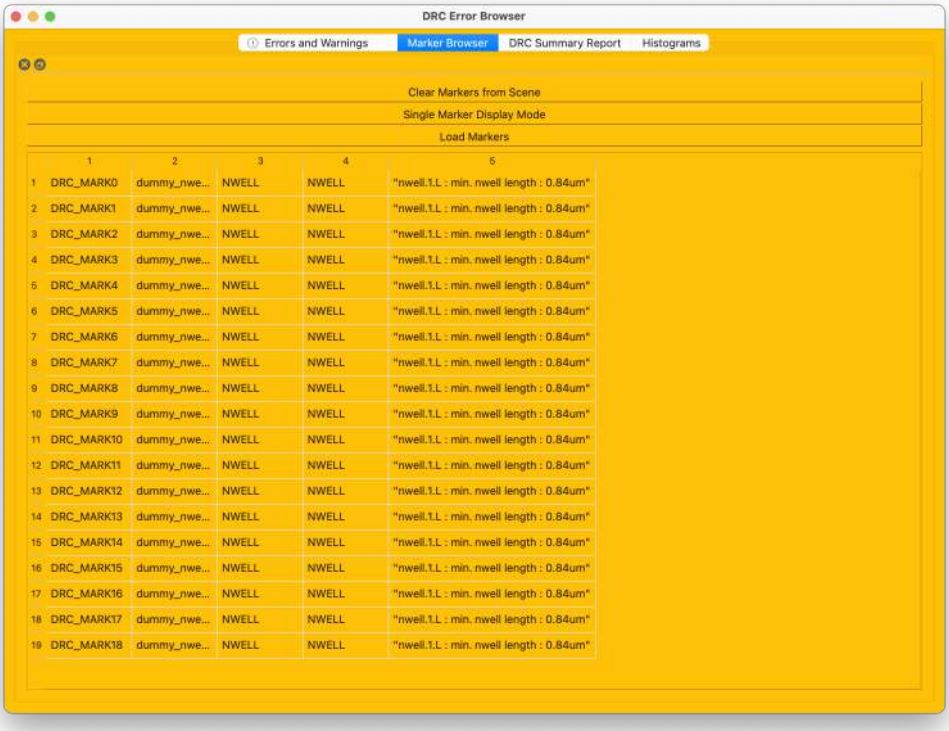


Figure 136: **Marker Browser** for `gcd1_DRC007`. Each row is a concrete violation instance of `dummy_nwell.1.L` on `NWELL`; columns show layer context and the textual rule caption.

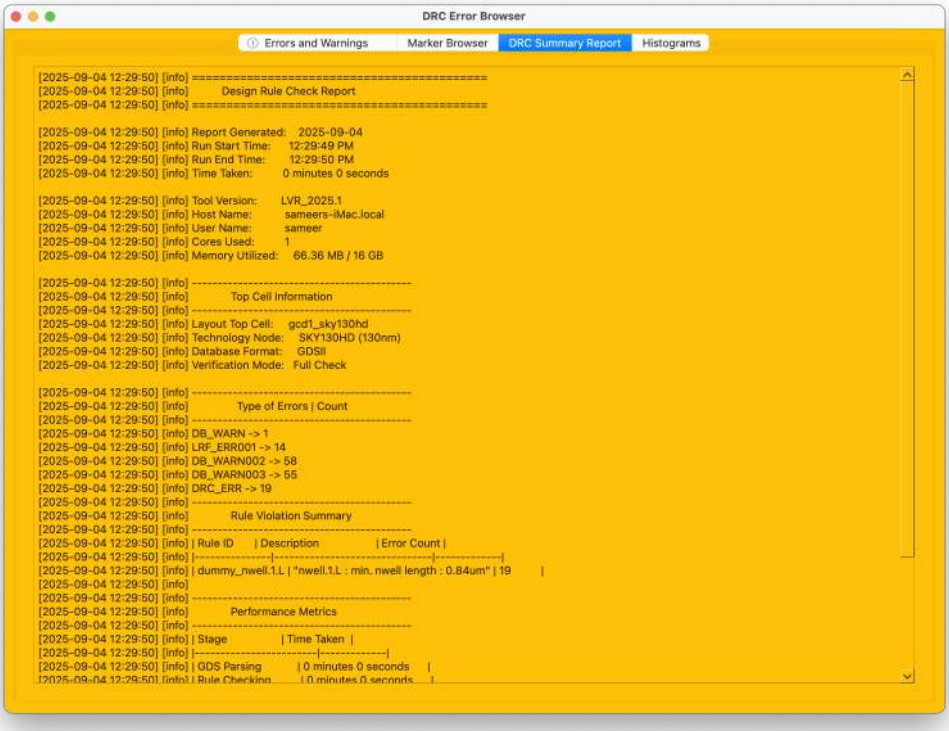


Figure 137: **DRC Summary Report**. Aggregated counts by rule; here the NWELL minimum edge-length rule dominates, confirming the injected condition is exercised by the layout.

Figure 138: **Histograms view**. The bar chart shows the distribution of violation measurements binned across the dataset; the heatmap provides a spatial tile density cue, helping you spot hot regions quickly.

clickable navigation to their locations in the viewer. Figure 137 summarizes the violation counts per rule ID (useful for regressions). Finally, Figure 138 visualizes how the violations are distributed in value (bars) and space (heatmap), which is handy for debugging pattern-based systemic issues in filler rows or channel regions.