RAMTERA TECHNICAL WHITE PAPER SERIES

WP-03

Proving a Layout Reader Is Lossless

Round-trip XOR as the qualification bar for GDSII and OASIS input/output

“It opens in the viewer” is not a correctness criterion. Zero-difference XOR against the source is.

12	50k – 1M	0	100%
DESIGNS VALIDATED	POLYGONS PER DESIGN	DIFFERING POLYGONS	ROUND-TRIP MATCH

Document	WP-03 — Proving a Layout Reader Is Lossless
Author	Sameer Pawanekar, Founder & Chief Executive Officer, Ramtera Inc.
Revision	v1.0
Date	August 2026
Classification	Public

ABSTRACT

Every physical verification result is downstream of the layout reader. A rule engine that is correct in every respect will still produce wrong answers if the geometry it was handed does not match the geometry on disk, and the failure is silent: markers appear in the wrong places, or fail to appear, with nothing in the log to indicate why. This paper argues that the only acceptable qualification for a layout input/output stack is a zero-difference XOR of a written-and-re-read database against its source, describes the specific failure classes that criterion catches, and reports the result across twelve designs ranging from fifty thousand to one million polygons.

01 The problem with informal validation

The usual test for a layout reader is that the design opens in a viewer and looks right. This is a weak criterion, and it is weak in a way that is easy to miss: it validates the parts of the reader a human notices, and silently accepts everything else.

A viewer will show a design that has lost a fill layer, mis-composed a nested transform by ninety degrees in one obscure cell, dropped every zero-area path, or silently rectilinearised a forty-five-degree edge. At a whole-chip zoom level all of these look identical to a correct read. The design still looks like a chip.

> **WHY THIS MATTERS MORE FOR A VERIFICATION TOOL THAN FOR A VIEWER**
> For a verification tool, the reader is not a convenience feature. It is the first stage of the measurement chain, and an error there is indistinguishable at the output from an error in the rule engine — except that it is far harder to find, because everyone debugging assumes the input is correct.

Nor is comparing polygon counts sufficient. Counts survive a transform composed in the wrong order, coordinates rounded in the wrong direction, and a boundary closed with a duplicated final vertex. Counting tells you nothing about position.

02 The criterion: zero-difference round-trip XOR

The test is straightforward to state. Read a layout database into the in-memory model. Write it back out. Read the written file. Compute a geometric XOR of the result against the source, layer by layer. Any polygon in the difference is a defect.

Stage	Operation	What a defect here indicates
1	Read source into the in-memory model	Parsing, record handling, hierarchy resolution
2	Write the model to a new file	Record emission, precision handling, hierarchy reconstruction
3	Read the written file back	Self-consistency of the reader/writer pair
4	XOR the result against the source, per layer	Any non-empty result is a positional or geometric discrepancy

Table 1 — The round-trip procedure.

> **WHY XOR RATHER THAN A STRUCTURAL DIFF**
> XOR is the right operator because it is positional and unforgiving. It does not care about polygon counts, vertex ordering, or how geometry is partitioned between polygons. It cares only whether the same area is covered on the same layer. Two databases that XOR to empty are the same layout, however differently they are structured internally.

The one weakness, and how to close it

A round trip validates the reader and writer *as a pair*. A symmetric defect — a transform composed wrongly on read and wrongly in the same way on write — cancels, and the XOR comes back empty on a database that is nonetheless wrong.

This is closed by breaking the symmetry: read the written file with an independent implementation, or read a file written by an independent implementation. Because the criterion is geometric rather than structural, a third-party tool that writes valid geometry in a completely different internal arrangement is a perfectly good source for the comparison.

03 What the criterion catches

The defects below are ones this test identified in our own implementation. Each was present in a reader that opened designs correctly in a viewer, and each would have produced silently wrong verification results.

Hierarchy transform composition

A cell placed inside a cell placed inside a cell accumulates a chain of transforms — translation, rotation, reflection, magnification. Composing that chain in the wrong order is correct for the identity case, correct for pure translation, correct for a single level of nesting, and wrong for a reflected instance inside a rotated one. In a large design that combination occurs in a handful of cells and nowhere else.

Structure collapse on repeated instances

Array references and repeated placements expand to many instances. An off-by-one in the expansion, or a bounding-region computation that collapses when a repetition has a degenerate axis, removes geometry that the viewer never draws attention to because the surrounding array is still there.

Rectilinearisation of angled geometry

A reader that converts every boundary to axis-aligned rectangles on ingest will handle the overwhelming majority of a digital design perfectly and destroy the forty-five-degree geometry in seal rings and crack stops. The round trip catches this immediately, because the written file no longer contains the diagonal edges the source had.

Precision and unit reconciliation

Database units and the declared precision determine how coordinates map to integers. A mismatch between the declared precision and the granularity actually present in the data produces geometry that is uniformly shifted or scaled — invisible at any zoom level, fatal for a spacing check.

Guard-ring and fill degeneracies

Long thin structures and dense fill arrays are where boundary-handling edge cases surface: zero-width elements, closed boundaries with a duplicated final vertex, self-touching rings. These are the shapes that most often expose an off-by-one in a boundary commit path.

04 Result

The criterion is applied across a corpus of twelve designs spanning approximately fifty thousand to one million polygons, covering digital blocks, mixed-signal blocks, and structures containing seal ring and fill geometry.

Measure	Result
Designs in the corpus	12
Polygon count range	~50,000 to ~1,000,000 per design
Designs achieving zero-difference round-trip XOR	**12 of 12**

Measure	Result
Polygons in the difference, all designs	**0**
Formats validated	GDSII (hierarchical read and write); OASIS (read and write)

Table 2 — Round-trip validation result across the corpus.

WHY THE TARGET IS ZERO AND NOT “CLOSE TO ZERO”
Zero is the only acceptable value here, and it is worth being explicit about why. A round trip that differs by one polygon in a million is not 99.9999% correct — it is a reader with a defect whose blast radius has not yet been established. The difference between zero and non-zero is categorical, not proportional.

05 Making the test operational

Run it in the writer

The most useful place for this check is not a separate test suite but inside the writer itself. After emitting a file, the writer can re-read its own output and XOR it against the model still in memory, refusing to report success if the difference is non-empty. The cost is one additional read and one boolean pass; the benefit is that a corrupt output file cannot leave the tool silently.

Run it per layer, not per design

A whole-design XOR tells you something is wrong. A per-layer XOR tells you which layer, which immediately implicates a datatype mapping, a specific record type, or a specific derivation. The additional cost is negligible and the debugging time saved is substantial.

Include the awkward designs deliberately

A corpus of clean digital blocks will pass a broken reader. The designs that earn their place in a round-trip corpus are the ones containing deep hierarchy with reflected and rotated instances, arrays with degenerate axes, angled seal-ring geometry, dense fill, and zero-area elements. If the corpus is comfortable, it is not testing anything.

Treat format conformance and fidelity as separate questions

A file can be perfectly conformant to the format specification and geometrically wrong, and it can be geometrically correct while using records a third-party tool will reject. Both need testing. The round-trip XOR answers the first; reading the output with independent tools answers the second.

Appendix A — Measurement environment

Component	Specification
Processor	Intel Core i7-13700K (13th generation, Raptor Lake)
Cores / threads	16 physical (8 performance + 8 efficient) / 24 logical
Memory	64 GB DDR5-4800 (2 × 32 GB), 1 GB swap
Storage	Kingston SNV2S500G NVMe SSD; all design data on NVMe
Operating system	Ubuntu 22.04.3 LTS (jammy), kernel 6.8.0-136-generic
Compiler	GCC 14.0.0 20231104 (experimental snapshot), `-O3`
Corpus	12 designs, approximately 50,000 to 1,000,000 polygons each
Comparison operator	Per-layer geometric XOR of round-tripped database against source
Acceptance criterion	Empty difference on every layer of every design

Table A1 — Measurement environment.

The corpus comprises designs available to Ramtera for validation purposes. Designs supplied under non-disclosure obligations are described here by scale only. The validation criterion and procedure are independent of the specific corpus and are reproducible against any layout database.

About Ramtera

Ramtera Inc. develops **LVR**, a physical verification engine that executes foundry SVRF rule decks natively, covering design rule checking, layout versus schematic, electrical rule checking, layout XOR, density and fill, and parasitic extraction. LVR is written in modern C++ with a Qt6 interface, and has been validated against publicly available process design kits including SKY130 and IHP SG13G2.

Ramtera also develops **PRE**, a place-and-route engine sharing LVR's geometry kernel, and maintains a portfolio of granted patents, registered copyrights and trademarks across its verification and implementation technologies.

Ramtera Inc. is incorporated in the United States, with research and development conducted by Ramtera Design Automation India Private Limited in Bangalore, India.

Contact	
Web	https://ramtera.com
General enquiries	info@ramtera.com
Technical support	support@ramtera.com
Series	Ramtera Technical White Paper Series
This document	WP-03, revision v1.0, August 2026