RAMTERA TECHNICAL WHITE PAPER SERIES

WP-05

One Database Unit

Precision, units and rounding semantics in cross-tool layout verification

Two tools can implement the same rule correctly and disagree, because they round in different directions.

1	2	∞	0
DATABASE UNIT	ROUNDING MODES	RESULTING DISAGREEMENTS	ACTUAL RULE ERRORS

Document	WP-05 — One Database Unit
Author	Sameer Pawanekar, Founder & Chief Executive Officer, Ramtera Inc.
Revision	v1.0
Date	August 2026
Classification	Public

ABSTRACT

Layout geometry is stored as integers. Design rules are written in microns. The conversion between the two involves a scale factor and a rounding decision, and that rounding decision is not specified by the rule language. Two verification tools implementing the same rule correctly can therefore disagree by one database unit, and a one-unit disagreement at a constraint boundary is the difference between a violation and a clean result. This paper sets out where the ambiguity lives, how it presents during correlation, and how to diagnose it — including a hypothesis we pursued at length and disproved, because the disproof is the more useful part.

01 Where the ambiguity lives

A layout database declares a user unit and a database unit. Coordinates are stored as integers in database units; the declared precision relates those integers to physical dimensions. A rule deck, meanwhile, expresses constraints in microns.

Executing the rule requires converting the constraint into the integer domain. That conversion is a division followed by a rounding step, and the rule language does not specify which rounding step. Truncation, rounding to nearest, and rounding away from zero are all defensible, and they differ whenever the constraint does not land exactly on a grid point.

WHY A ONE-UNIT DIFFERENCE IS NOT A SMALL DIFFERENCE

The disagreement is always at most one database unit. That sounds negligible, and it is negligible everywhere except at a constraint boundary — which is precisely where every design rule check does its work. A spacing of exactly the minimum is either a violation or it is not, and one unit decides which.

Where the geometry meets the same problem

The constraint side is only half of it. Derived geometry — the result of sizing, boolean operations, or projections — also lands on non-integer coordinates that must be resolved to the grid. A sizing operation of half a grid unit has to go somewhere, and whether it goes outward or inward changes whether the derived layer touches its neighbour.

02 How it presents during correlation

The signature is distinctive once recognised, and misleading before that.

Observation	What it suggests	What it usually is
A rule differs by a handful of markers, on a rule with thousands	A subtle geometric bug	Rounding at the constraint boundary
The differing markers are all at exactly the constraint value	Coincidence	Confirmation — this is the signature
A whole family of related rules moves together	One shared derivation is wrong	One shared rounding site in the operator, not the rules
Every coordinate is off by a constant factor	A rounding issue	**Not** rounding — a unit or precision mismatch, which is a different fault

Table 1 — Distinguishing rounding disagreements from their neighbours.

THE DISTINCTION THAT MATTERS MOST

The fourth row is the important one. A uniform scale error and a rounding disagreement produce superficially similar symptoms — markers in almost the right place — and have completely different causes. Confusing them costs days.

03 A hypothesis we pursued and disproved

During a correlation study we observed a systematic discrepancy and formed a hypothesis: that the database unit declared in the layout and the precision recorded in the reference tool's result database differed by a factor of ten, and that every coordinate was consequently scaled.

The hypothesis was coherent, it explained the observed magnitudes, and it was wrong. It was disproved by an independent bounding-box measurement: reading the design with a third-party layout tool and comparing the reported extents against both the layout database and the reference results. The extents agreed. There was no scale factor.

> **WHAT IT ACTUALLY WAS**
> The actual cause was mundane — a parser dropping one record per rule because it did not handle a cell-context record type in the reference database format. One dropped rectangle per rule, across many rules, produced a systematic-looking deficit that a scale hypothesis appeared to explain.

The general lesson is worth more than the specific one. A hypothesis that explains the magnitude of a discrepancy is not thereby supported; magnitudes are easy to match and hard to distinguish. What settles the question is an independent measurement that the hypothesis makes a specific prediction about. A bounding-box comparison against a third tool costs minutes and either confirms or kills a scale hypothesis outright.

Diagnostic ordering

1. **Check extents first.** Read the design with an independent tool and compare bounding boxes against both the layout database and the reference results. This settles the unit and precision question before any rule-level analysis begins.
2. **Then check whether the differing markers cluster at the constraint value.** If they do, the cause is rounding. If they are scattered across the constraint range, it is not.
3. **Then check whether related rules move together.** A family moving as one implicates a shared operator or derivation, not the individual rules.
4. **Only then look at the rule implementation.** By this point the search space is small.

04 Inferring the true granularity

A layout database declares its precision, but the declaration can be inconsistent with the data. A useful and cheap consistency check is to infer the actual granularity from the data itself: compute the greatest common divisor of the coordinate values present and compare it against the declared precision.

A GCD substantially coarser than the declared precision indicates the data is on a coarser grid than claimed — which is not necessarily an error, but is always worth knowing before comparing against another tool's output. The check costs one pass over the coordinates and can be run at load time.

05 Recommendations

For tool implementers

1. **Round in one place.** Every micron-to-unit conversion should pass through a single function. Scattered conversion sites will not agree with each other, let alone with another tool.
2. **Decide deliberately, and record the decision.** Truncation and round-to-nearest are both defensible; being inconsistent between them is not. The choice belongs in documentation, not only in the source.

3. **Validate declared precision against observed granularity at load time.** A cheap check that catches an entire class of problems before they become rule-level mysteries.
4. **Preserve the pre-rounding value where a rule needs it.** Some constraints are naturally expressed against the exact value; rounding early discards information that cannot be recovered.

For anyone running a correlation study

1. **Establish unit and precision agreement before comparing any rule.** A correlation study run across a unit mismatch produces a table of numbers that describes nothing.
2. **Treat one-unit boundary disagreements as a distinct category.** They are neither tool defects nor noise; they are an unspecified behaviour in the rule language on which two implementations have made different, defensible choices.
3. **Report them separately from genuine mismatches.** Folding them into an error count misrepresents both the tool and the reference.

06 Conclusion

Rounding is the least interesting source of disagreement between two verification tools and one of the most common. It produces small, systematic, boundary-clustered differences that resemble subtle geometric bugs and are nothing of the kind.

The practical defence is ordering: settle units and precision with an independent extent measurement before examining any rule, then look for boundary clustering, then look at families, and only then at individual implementations. Each step is cheap and eliminates a large part of the search space. Reversing the order — starting from the rule and working outward — is how a one-unit rounding difference consumes a week.

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-05, revision v1.0, August 2026