Semiconductor Technology from A to Z

Digital technology

Layout versus Schematic (LVS)

Introduction & Motivation

Why DRC Alone Is Not Enough

A layout that fully passes the Design Rule Check is geometrically sound: every width, spacing, enclosure, and density falls within the allowed limits. That, however, says nothing about whether the drawn polygons actually implement the circuit designed in the schematic. A missing contact, two swapped terminals, or an accidentally duplicated transistor violate no design rule at all — the layout stays "DRC clean" even though the circuit no longer matches the design electrically.

Such errors happen easily: during manual routing of a complex block, while copying and adapting existing layout cells, or simply through a forgotten connection step. The larger the chip, the more unlikely it becomes to spot a single missing contact by eye among millions of structures.

The Idea Behind LVS

The Layout-versus-Schematic comparison (LVS) solves this problem by checking not the geometry itself but the circuit topology that results from it. A netlist is automatically extracted from the physical layout — a list of all devices (transistors, resistors, capacitors) together with their terminals and the nets connecting them. This extracted netlist is then compared against a reference netlist derived directly from the schematic.

If the two netlists match — the same devices, the same connections, the same device parameters — the layout is considered "LVS clean." Any deviation is reported as a mismatch and must be resolved before the next step in the design flow. LVS is therefore the necessary complement to DRC: DRC ensures the layout is manufacturable; LVS ensures that what gets manufactured is actually the intended circuit.

Netlist Extraction from the Layout

From Polygons to Devices

The first step of LVS extraction identifies devices purely from layer geometry — much like a DRC tool, just with a different goal. Wherever a polysilicon structure overlaps a diffusion area, the extractor recognizes a transistor: the width of the

polysilicon at the overlap yields the gate width W, and the extent of the diffusion perpendicular to it yields the gate length L. Whether the device is NMOS or PMOS follows from the diffusion type (N^+ or P^+), or more precisely, from whether the diffusion sits inside an N-well.

Other devices are recognized analogously: a metal-insulator-metal structure yields a capacitor; a narrow, high-resistance polysilicon or diffusion strip with no gate function yields a resistor. Every device found is added to the netlist together with its extracted geometric parameters.

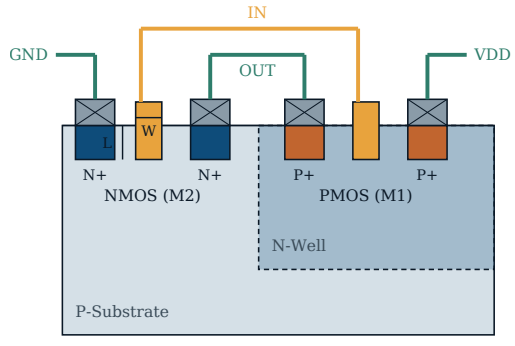
From Contacts to Nets

In parallel, the extractor determines electrical connectivity: all polygons on the same layer that directly touch or overlap, as well as all layers connected to each other through contacts or vias, are merged into a common net — regardless of how many metal layers and vias that net spans. Technically, this comes down at its core to a union-find operation over all conductive polygons: two polygons belong to the same net exactly when a conductive path exists between them.

The result is a complete netlist: a list of all devices with their terminals, where each terminal is assigned to one of the previously determined nets. At this stage, the netlist still has no names like "VDD" or "OUT" — it references nets only through internal identifiers. Only the comparison against the schematic (Section 3) assigns them meaningful names again.

The diagram below shows a simple inverter in layout and the simplified netlist extracted from it.

Netlist Extraction: Inverter Layout




```
* extracted netlist

M1 OUT IN VDD VDD
   PMOS W=0.60u L=0.35u

M2 OUT IN GND GND
   NMOS W=0.40u L=0.35u
```

SPICE-like notation

The Reference Netlist from the Schematic

From Schematic to Netlist

While the layout netlist is derived from geometry, the reference netlist is obtained directly: a schematic capture tool already knows, for every placed symbol — transistor, resistor, capacitor — its terminals (pins) and their meaning. When the designer connects two pins with a drawn wire or with identical net labels, they belong to the same net. This extraction is trivial compared to the layout side, since no geometry needs to be interpreted — the connectivity is already explicit.

Net names such as VDD, GND, IN, or OUT are usually assigned by the designer themselves, either by labeling individual wires or through special global net symbols for supply rails that are treated as "the same everywhere" across the tool — a VDD symbol placed at several points in the schematic connects all of those points into a single net, even without a continuous wire between them.

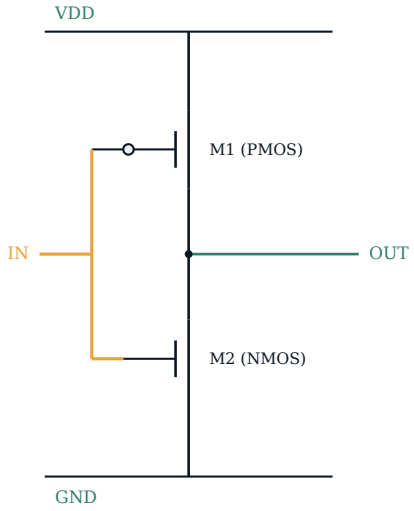
Hierarchy and Flattening

Real schematics are almost always built hierarchically: an inverter is a cell reused inside a gate, which is itself part of a multiplexer instantiated inside a register. The netlist compiler must resolve this hierarchy — either fully flat (every instance of a cell is expanded into its own independent devices) or hierarchically, where identically structured instances are checked only once against a reference cell, after which merely their interconnection with one another is compared.

Hierarchical LVS is the only practical approach for memory blocks with thousands of repeated, identical cells (e.g. SRAM arrays) — a fully flattened comparison would recompute the entire cell structure for every single instance, even though only its position in the layout differs.

The reference netlist obtained this way is considered "golden" — the authoritative target state against which the netlist extracted from the layout is compared in the next step. The diagram below shows the schematic of the same inverter from Section 2 and the reference netlist derived from it.

Reference Netlist: Inverter Schematic




```
* reference netlist
M1 OUT IN VDD VDD
   PMOS W=0.60u L=0.35u
M2 OUT IN GND GND
   NMOS W=0.40u L=0.35u
```

golden — the authoritative target

The Comparison Algorithm

The Netlist as a Graph

To compare two netlists algorithmically, each is first converted into a graph: devices and nets become nodes, and every terminal connection of a device to a net becomes an edge between them, labeled with the terminal role (drain, gate, source, bulk). The result is a so-called bipartite graph — device nodes connect only to net nodes, never directly to one another.

Two netlists are electrically identical exactly when an isomorphism exists between their two graphs: a unique mapping that sends every node of one graph to exactly one node of the other such that edges, edge labels, and node attributes (device type, W/L) are preserved. Graph isomorphism is computationally expensive in the general case — for circuits with millions of transistors, a naive brute-force comparison of all possible mappings would be hopeless.

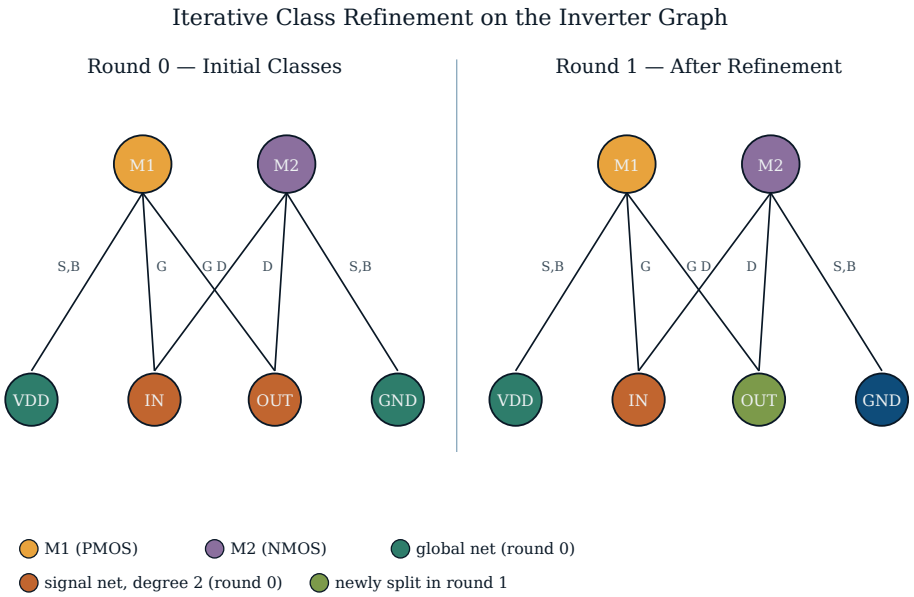
Iterative Refinement Instead of Brute Force

LVS tools sidestep this problem through iterative class refinement. Every node first receives a coarse initial class based on easily determined features — device type and parameters for device nodes, number of connected terminals for net nodes. The algorithm then repeatedly refines: each node is assigned a new, finer class derived from the multiset of its neighbors' classes (together with the edge labels). Two nodes initially placed in the same class split apart as soon as their

neighborhoods differ.

This process repeats until no class can be split any further. In well-designed circuits with little symmetry, what typically remains are almost exclusively singleton classes — each node is then uniquely matched to a node in the other netlist. Only for the few remaining, genuinely symmetric groups (for example, parallel identical dummy fingers of a transistor) does the algorithm still need to try out all remaining assignment possibilities within that small group — an effort that stays practical even with millions of devices, because these residual groups are, in practice, tiny.

The diagram below shows this refinement step using the inverter from the previous sections: initial classes on the left, the result after one refinement round on the right.



Typical Mismatches

Four Recurring Failure Patterns

Although LVS tools produce very different error messages in detail, the vast majority of mismatches trace back to a handful of recurring causes. Knowing these patterns usually finds the root cause in the layout far faster than working blindly through the error list.

An open net results from a missing or misplaced contact: two structures that should be connected according to the schematic remain electrically separate in the layout. The netlist then shows two nets instead of one — the LVS typically

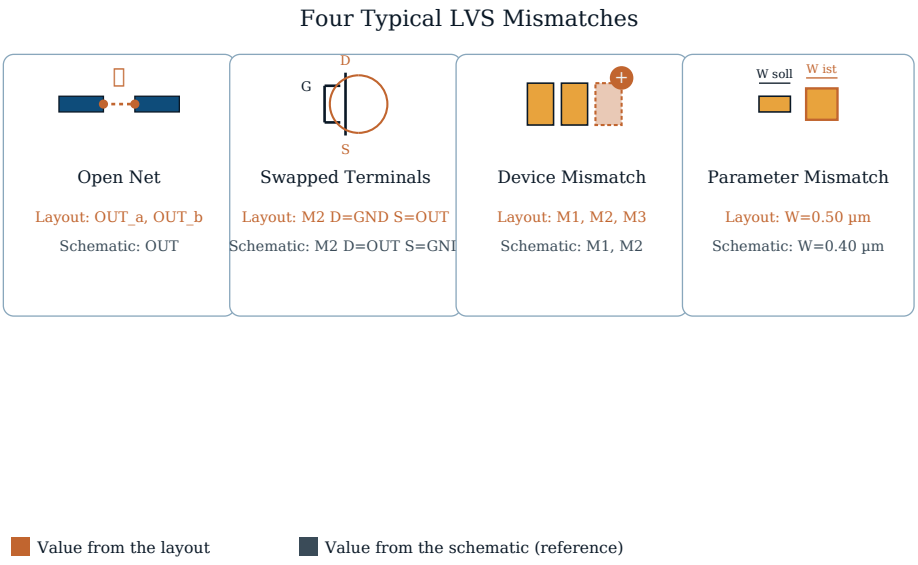
reports a "net not found" or a net-count mismatch, because a net expected from the schematic falls apart into two pieces in the layout.

Swapped terminals happen easily when copying or manually wiring a transistor: source and drain are connected to the wrong nets in the layout. Since source and drain are geometrically symmetric — and therefore interchangeable — for most MOSFETs, the layout stays DRC clean, but the circuit then behaves electrically differently than intended, or not at all.

A missing or extra device usually results from a copy error — a cell was accidentally placed twice, or removing a test structure also deleted a structure that was actually needed. The two netlists then already differ in the sheer device count, often the simplest of all mismatches to find.

A parameter mismatch usually concerns transistor width W or length L: if the layout was drawn with a different gate width than the schematic specifies, connectivity remains fully correct — topology and device type match, only the numeric value deviates. Many LVS tools do not flag this as a hard error but as a warning with an adjustable tolerance, since small deviations from rounding to the manufacturing grid are often unavoidable.

The diagram below shows all four failure patterns side by side, layout versus schematic.



LVS in the Design Flow

Position Between DRC and Parasitic Extraction

In the design flow, LVS fundamentally runs after the Design Rule Check: a

layout must first be DRC clean before an LVS run can produce meaningful results — on a geometrically flawed layout (for instance with structures overlapping that should actually be separate), no reliable netlist could be extracted in the first place. Together, both checks form what is referred to in practice as "signoff": a layout that is neither DRC clean nor LVS clean does not proceed to fabrication.

As with DRC, an LVS run is not a one-time event at the end but part of an iterative cycle: every reported deviation — open net, swapped terminal, missing device — must be fixed in the layout, after which extraction and comparison run again. For large blocks, incremental LVS runs that, after a small layout change, only re-compare the affected neighborhood considerably speed up this cycle compared to a full re-comparison of the entire block.

What Follows Next

Once a block is both DRC clean and LVS clean, parasitic extraction usually follows: from the now-verified layout geometry, the parasitic resistances and capacitances of the wiring are additionally computed and added to the netlist — a task LVS itself does not perform, since it checks only connectivity and device parameters, not the electrical fine details of the wiring. The resulting parasitics-annotated netlist then feeds post-layout simulation, which shows whether the circuit still behaves as intended once the real, layout-induced delays and voltage drops are taken into account.

Only once this final step also confirms the specification is a block ready for tape-out — the release of the mask data to the foundry.

The diagram below shows this flow at a glance, from schematic and layout through to tape-out.

LVS in the Design Flow