Aniah
Zero Electrical Error on 1st Silicon Spin

WHITE PAPER 2026

CONTEXT IS **EVERYTHING**

WHY AI. FOR ANALOG DESIGN REQUIRES MORE THAN YOUR NETLIST

1.

EXECUTIVE SUMMARY

AI is entering Electronic Design Automation. Large Language Models can understand technical language, reason through problems, and generate explanations.

The obvious question: can you feed a netlist to an LLM and get useful verification results?
The problem: a netlist describes *what* is connected, not *why*. Design intent, power domain semantics, operating modes, system constraints exist in specs, PDK documentation, and the designer's head. Not in the netlist.

LLMs have seen the textbooks, but pattern-matching against training data isn't verification. They can produce plausible-sounding explanations. But plausible isn't enough for sign-off.

The real question: can you audit and trust the output at tapeout?

We argue that *meaning* must be reconstructed from the netlist before AI can help. Transform raw structure into a semantic model that captures electrical roles, power domains, signal paths, constraints, and causality. Then apply AI to that.

Aniah OneCheck® does this reconstruction, performing a fast, deterministic electrical analysis that builds a semantic model from the netlist. Amigo®, Aniah's AI assistant, operates on this verified model, not on raw netlist text.

AI doesn't replace analysis or discover new failure modes. It helps engineers navigate results, understand violations, and decide what to fix.

For AI to be trustworthy in analog verification, context must be constructed deterministically before AI is applied.

This isn't optional. **It's foundational.**

2.

INTRODUCTION AND SCOPE

THE SCALE PROBLEM

LLMs have context limits. A design with hundreds of thousands of devices already pushes those limits. Industrial mixed-signal SoCs contain millions or billions of devices. The state space grows combinatorially: a design with 50 control bits has 2^{50} possible configurations. You can't enumerate them.

A useful representation must be small enough to process but rich enough to reason about. Dumping more netlist into the prompt doesn't help, the reduction has to preserve what matters.

ARCHITECTURE OVERVIEW

OneCheck performs fast, deterministic electrical analysis that builds a semantic representation from the netlist. Two mechanisms make this tractable:

- Clusterization identifies repeated structures and electrical patterns, then factorizes them. Violations sharing a common root cause collapse into single issues. In a production 65nm mixed-signal design, 11,847 violations clustered into 53 root causes. Larger designs cluster better: **complexity doesn't scale linearly with device count**.

- System Conditional Analysis (SysCon) determines which states are actually reachable. Instead of enumerating all configurations, it works backward from a violation to find the minimal conditions that trigger it. Structurally possible but functionally unreachable violations get filtered out before any AI interaction.

Amigo operates on this verified semantic model through MCP (Model Context Protocol). It receives curated context for each violation, including the rule violated, the relevant netlist fragments, power scenarios, propagation paths, PDK constraints, and causal chains.

EXPLICIT TRADE-OFFS

This approach makes deliberate choices:

Determinism over discovery
OneCheck detects problems covered by its rule set and PDK integration. Nothing outside that scope. It does not discover entirely new failure modes. We trade broad discovery for predictability and sign-off compatibility.

Explanation over autonomy
Amigo reasons within bounds established by OneCheck. It cannot override analysis results, invent operating assumptions, or explore unbounded state spaces. This constraint ensures that explanations remain reproducible and auditable.

Architecture over benchmarks
This paper covers how the system works, not runtime performance metrics. Quantitative evaluation is future work.

This paper covers how LLMs operate on circuit data, why raw netlists provide insufficient context, and how semantic reconstruction enables reliable AI interaction. It describes clusterization, system conditional analysis, and Amigo's architecture.

It does not cover runtime benchmarks, comparison studies against alternative AI approaches, detailed MCP implementation, PDK integration methodology, or formal correctness proofs.

This is a position paper describing an architecture, not a product datasheet or academic benchmark study.

3.

Terminology and definition

For clarity, the following terms are used throughout this paper with the meanings defined below.

Netlist
A hierarchical description of transistors and devices connected by nets. May include parasitic resistors and capacitors from extraction.

Mixed-signal verification
The verification of designs that combine analog and digital circuitry, including the validation of electrical correctness, power intent, domain interactions, and behavior across operating modes and configurations.

Formal analysis
Vectorless analysis that reasons symbolically about all possible electrical configurations to prove compliance with predefined design rules.

Propagation path
An electrically conductive path through devices and interconnect from a supply rail to another rail or to ground, along which current can potentially flow under certain conditions.

Interval modelling
A modeling technique in which node voltages (and possibly currents) are represented as bounded ranges (minimum and maximum values) rather than precise numerical values, in order to enable scalable analysis.

State analysis
Exploration of reachable circuit states, as would be attained by Spice DC analysis using suitable testbenches. Voltage and current modelling is adjusted for scalability.

Transistor-level
A representation of an Integrated Circuit (IC) as an electrical schematic of wires, transistors and other devices. Usually in a Netlist format; necessary for LVS sign-off.

Semantic model
A structured representation capturing not just connectivity but electrical roles, power domain relationships, operating conditions, and causal dependencies.

4. THE RISE OF AI IN ELECTRONIC DESIGN AUTOMATION

4.1 WHY AI IS APPEALING FOR ANALOG DESIGN

Analog and mixed-signal design is hard and getting harder. Complexity grows with each node. Power domains multiply. Reliability requirements tighten. Meanwhile, schedules compress and first-pass silicon success remains non-negotiable.

LLMs can parse documentation, generate explanations, and handle follow-up. This is very useful for understanding error reports and tracing root causes.

4.2 HOW LARGE LANGUAGE MODELS OPERATE

LLMs operate on statistical patterns, not physics. They can parse netlist syntax but can't evaluate whether a current path is realizable or a state is reachable under realistic conditions.

They can synthesize and explain, not through explicit rules, but through learned statistical structure.

LLMs extract intent from well-organized text but have no access to physical or causal processes. Any reasoning about circuit validity depends entirely on what's in the input.

This constrains their application to circuit analysis. A raw SPICE or CDL netlist is syntactic: devices and connections. It doesn't encode design intent, power semantics, operating modes, or system-level causality. An LLM can parse this text, but the information needed to reason correctly isn't always there.

The model produces plausible explanations by extrapolating from training patterns, not from analysis of this circuit.

LLMs work well on structured semantic representations. The limiting factor isn't the model, it's the input.

4.3 ALTERNATIVE AI APPROACHES

Multiple AI approaches target electronic design, each with different objectives. This section reviews the major directions and clarifies how our architecture differs.

GRAPH-BASED AND TOPOLOGY-LEARNING MODELS

Graph Neural Networks (GNNs) like CircuitGNN, DeepGate, or HOGA, learn circuit behavior from structural representations. They treat netlists as graphs (devices as nodes, connections as edges) and capture topological patterns. Results are strong for performance prediction, circuit classification, and parameter estimation.

GNNs are statistical and predictive. They can predict a circuit is "likely" to have a timing violation, but it can't explain why or guarantee correctness when the prediction is negative. No audit trail. For sign-off, predictive confidence isn't enough.

DOMAIN-ADAPTED LANGUAGE MODELS

Another direction is adapting LLMs to chip design through domain-specific techniques. NVIDIA's ChipNeMo applies custom tokenizers, domain-adaptive continued pretraining, supervised fine-tuning, and retrieval models to create design-specialized assistants. Their evaluated applications include engineering chatbots, EDA script generation, and bug summarization.

This approach works well for tasks that benefit from design knowledge accumulated across projects, like methodology guidance or pattern recognition in bug reports. The model learns what's generally true about chip design.

Our focus is different. We're reasoning about a specific circuit instance, this netlist, these power domains, these operating conditions. ChipNeMo helps an engineer understand what to do in general. OneCheck and Amigo verify whether this specific design does it correctly.

The approaches are complementary. Domain-adapted models provide methodology expertise; instance-grounded analysis provides verification results. Both have a role in a complete design flow.

SIMULATION-IN-THE-LOOP AI SYSTEMS

A powerful paradigm combines AI with circuit simulation. The AI proposes; the simulator validates. Models invoke SPICE to test hypotheses, explore operating points, and validate candidate fixes.For small to moderate circuits (tens of thousands of devices), this works well. But simulation is instance-based and scenario-based. Each run covers one configuration, one input set, one operating point.

The complexity makes exhaustive coverage infeasible regardless of simulation speed. Simulation-in-the-loop is valuable for deep debug after issues are found, but it doesn't replace formal verification for completeness.

FORMAL VERIFICATION COMBINED WITH LEARNING HEURISTICS

Academic research combines SAT/SMT solvers with learned heuristics like NeuroSAT or Graph-Q-SAT, to accelerate formal verification. ML guides variable ordering, clause selection, or proof search, achieving orders-of-magnitude speedup on specific problem classes.

Our work targets a different layer: constructing human-usable representations of results and enabling meaningful interaction. Both directions use ML to enhance formal methods. They're compatible and could combine.

POSITIONING

The optimization target differs from exploratory AI. The right question is not “what might be wrong?” but “what is wrong, why, and can I trust this answer at tapeout”?

COMPARISON SUMMARY

Approach	Primary Strength	Limitation for Sign-Off Verification	Relationship to This Work
GNNs / Topology Learning	Fast estimation, pattern recognition	No causal explanation; not auditable	Useful for early exploration
Fine-Tuned Domain LLMs	Documentation search, methodology assistance	No grounding in specific design instance	Useful for general assistance
Simulation-in-the-Loop	Validates specific scenarios with high fidelity	Coverage limited to simulated cases	Useful for targeted debug
Formal + ML Heuristics	Accelerates solver performance	Targets speed, not explainability	Compatible philosophy; potential integration point
This Work (OneCheck + Amigo)	Deterministic, auditable, causally grounded	Depends on rule coverage and PDK integration	–

5. THE FUNDAMENTAL PROBLEM: THE LACK OF CONTEXT

5.1 A NETLIST IS NOT THE CIRCUIT

SPICE and CDL netlists are essential for simulation, extraction, and sign-off. Every verification flow depends on them. They precisely describe transistor-level structure: devices with their models and parameters, nets with their connectivity, hierarchy with its instantiation. For what they're designed to do (feed simulators, enable LVS, support extraction) they work extremely well.

But netlists are not semantic models. They describe what is connected, not why, not under which conditions, not with what intent.

A device line says *"here is an NMOS with these dimensions connected to these nets."* It doesn't say *"this NMOS is part of an ESD clamp that only activates during surge events."* A net definition says *"this wire connects these terminals."* It doesn't say *"this is a power-down control signal that must settle before the core domain enables."* The information that matters for correctness like the design intent or the operating context exists somewhere, but not in the netlist.

A netlist is syntax, not semantics. It accurately describes connectivity but doesn't capture:

- **Power domain relationships and voltage intent**
  Whether supplies are meant to be always-on, mutually exclusive, sequenced, or conditionally enabled is not encoded in connectivity alone.

- **Functional roles of signals**
  Whether a net is a data signal, a control signal, a bias, a test hook, or a protection path cannot be deduced reliably from structure.

- **Conditional behavior and operating modes**
  Many circuit regions are only active under specific configurations, power states, or control conditions.

- **System-level interactions across hierarchy**
  A block's architectural role and its interaction with the rest of the system aren't explicit at the netlist level.

Two circuits with different purposes can look identical in a netlist. A single architectural issue can scatter across thousands of devices. Conversely, a locally suspicious structure may be perfectly valid in context.

This isn't a flaw in SPICE or CDL, it's their purpose. Netlists were designed to describe how a circuit is built, not what it's meant to do. Reasoning about correctness requires that semantic layer to be reconstructed from other sources: specifications, methodology documents, PDK rules, and structured analysis. The netlist is the starting point, not the answer.

PRACTICAL EXAMPLE A: **THE DIODE-CONNECTED MOS**

THE STRUCTURE

A MOS transistor with gate tied to its drain, commonly called a "diode-connected MOS", appears in the netlist as:

```
M1 D D S B nmos
```

Structurally, this is a normal device. No syntax error, no illegal connection, no obvious violation.

THE AMBIGUITY

This identical structure serve fundamentally different purposes:

Context	**Function**	**Validity**
Between I/O pad and VSS (ground)	ESD protection clamp	**Correct** – shunts surge current
In a bias network	Current-to-voltage converter	**Correct** – generates stable VBIAS
In a core logic data path	Unintentional DC path	**Error** – causes static current
Between two power domains	Level shifting element	**Conditional** – depends on voltage relationship

CONCRETE SCENARIO

Consider `M1 D D S B` nmos connected between I/O pad and VSS (ground):

In an ESD clamp: The diode-connected MOS shunts surge current to VSS (ground) during an ESD event. The device is specifically sized for high transient current and uses thick gate oxide. This is correct by design.

In a core logic path: The same topology creates a permanent DC path to VSS. Current flows whenever the node voltage rises above Vth. This causes static power consumption, potential voltage contention, and in severe cases, latch-up. This is a critical error.

WHAT IT DEMONSTRATES

The netlist line `M1 D D S B` nmos is identical in both cases. The difference is:

- Which net the drain connects to
- What power domain the device sits in
- What voltage and current conditions it experiences
- What the designer intended

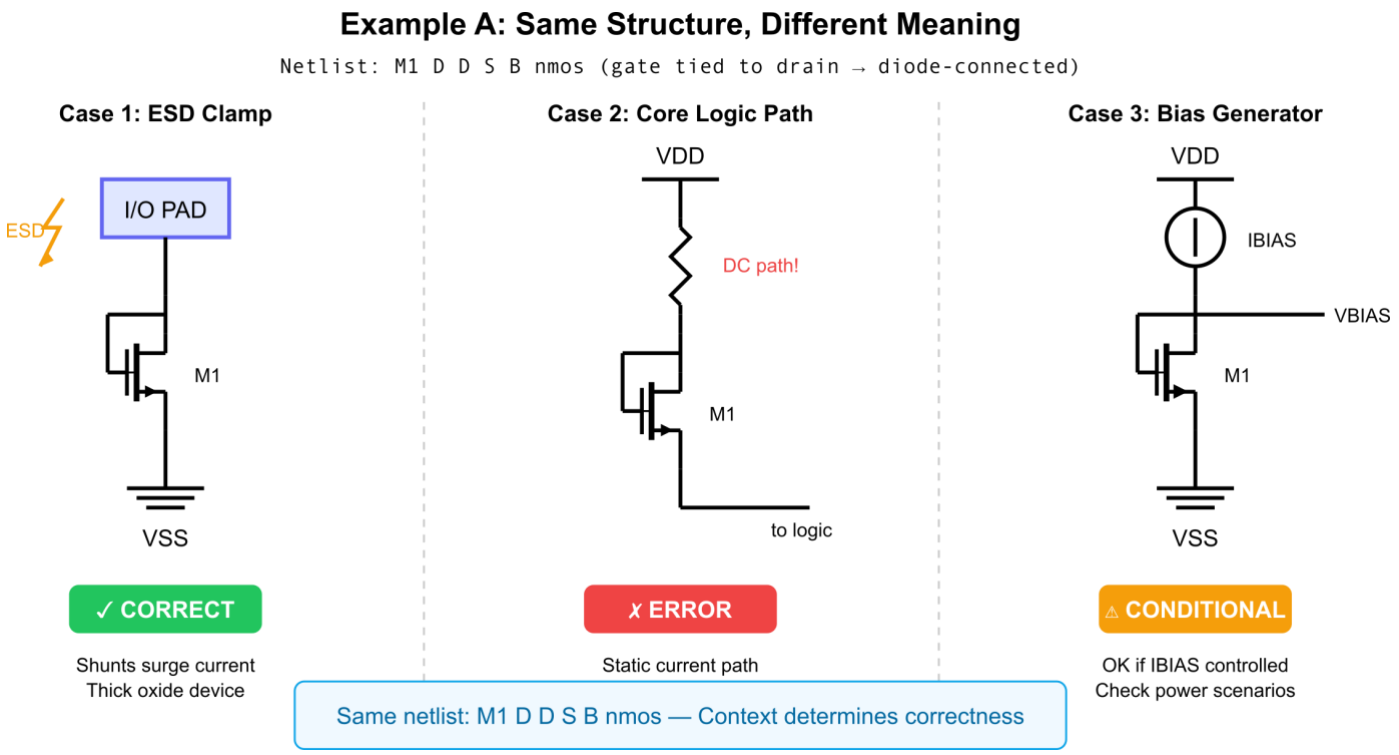

Figure 1 Example A - Diode-Connected MOS

5.2 WHY A NETLIST ALONE IS NOT SUITABLE FOR AI REASONING

Feeding a raw netlist to an LLM and asking it to "find errors" assumes the model can reconstruct semantic layers from structure alone, infering power intent, operating modes, signal roles, architectural boundaries, causality. It can't. Not because it's incapable, but because the information isn't there.

Faced with such input, the model falls back on statistical patterns from training; not verifiable understanding of this specific circuit.

Four failure modes result:

Ambiguity is unavoidable. Without power intent and operating scenarios, the model can't determine if a situation is acceptable, forbidden, or irrelevant.

Plausible but ungrounded explanations. LLMs produce coherent text. They may generate explanations that sound correct but aren't anchored in actual analysis of this circuit.

Inconsistent reasoning. Without stable root causes grounded in electrical causality, similar situations get explained differently. Multiple manifestations of the same issue get treated as unrelated.

Scalability collapse. Larger netlists add syntactic detail without semantic clarity. The model drowns in low-level information while lacking abstractions for reasoning.

Structurally complete. Semantically insufficient.

PRACTICAL EXAMPLE B: **THE THIN-OXIDE OVERSTRESS**

THE STRUCTURE

A thin-oxide NMOS device in an inverter:

```
M1 OUT IN VDD VSS nmos_1p2    // 1.2nm oxide, Vgs_max = 1.1V
```

- All terminals are connected correctly
- No shorts, no opens, no illegal topology

THE PROBLEM

The netlist is structurally valid. But electrical validity depends on operating scenarios:

Scenario	VDD	IN Swing	Vgs_max	Result
Normal operation	1.0V	0 → 1.0V	1.0V	**OK** (within 1.1V rating)
High-voltage mode	1.2V	0 → 1.2V	1.2V	**Overstress** (+9% over rating)
Test mode (ATE)	1.0V	0 → 1.8V	1.8V	**Catastrophic** (+64% over rating)

WHAT IT DEMONSTRATES

The circuit passes all structural checks. The error only exists when you combine:

- Device ratings (Vgs_max = 1.1V from PDK)
- Operating scenarios (which supply voltages are active)
- Signal conditions (what voltage swing the input sees)

Without ratings, scenarios, and reachable states, the netlist looks correct.

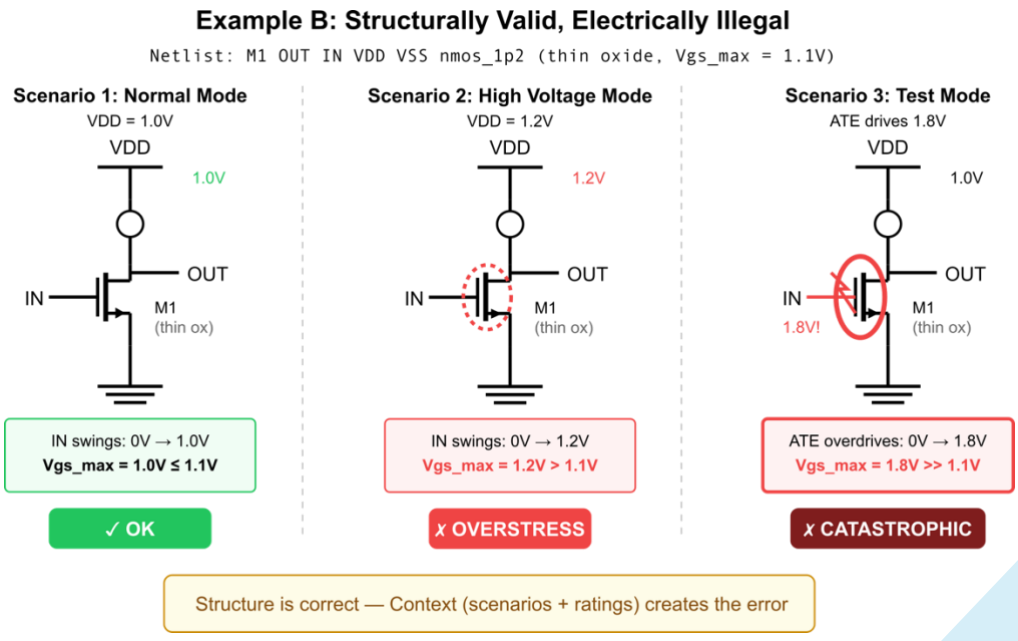

Figure 2 Example B - Structuraly Valid, Electrically Illegal

5.3 THE REPRESENTATION WALL AND THE STATE-SPACE WALL

One response is to enrich the input by adding power scenarios, operating modes, device ratings. If the netlist lacks semantic information, providing it explicitly. This moves in the right direction but it doesn't solve the problem. It transforms one problem into two others.

The key issue is not what the LLM does internally, but what representation must be provided for reliable reasoning. And once you start making semantic information explicit, two independent forms of explosion appear.

Representation explosion. A raw netlist is syntactically compact but semantically incomplete. Now start adding the information needed for correct reasoning such as power domain definitions, operating mode descriptions, voltage levels for each scenario, device ratings from the PDK, topology classifications, signal roles, hierarchy annotations. Each layer multiplies the description size. What was a compact structural file becomes an enormous knowledge base, quickly exceeding any practical context window.

State space explosion. Correctness in analog and mixed-signal circuits is conditional. A circuit isn't just correct or incorrect, it's correct under certain conditions and potentially incorrect under others.

A modest control structure like 10 registers of 8 bits, represents 80 bits of state: $2^{80} \approx 1.2 \times 10^{24}$ possible configurations. You can't describe them textually. You can't ask an LLM to reason about "all possible configurations" because there's no way to present them.

These walls are independent. Compress the structural representation and conditional behavior remains combinatorial. Restrict the state space and structural context remains too large. Solving one doesn't help with the other.

This isn't a limitation of current LLMs. It's a fundamental constraint on any approach presenting raw or weakly structured design data to a text-based reasoning system.

Larger context windows shift the wall; they don't remove it. The growth in required information is driven by design complexity, not model limits.

6. FROM SYNTAX TO SEMANTICS

6.1 THE GAP BETWEEN STRUCTURE AND MEANING

Designers don't think in netlists. They think in power domains, functional roles, operating modes. Ask any designer to describe their circuit and they'll say “the bias network” or “the isolation boundary”. Not “M1437 connected to net2891”.

Netlists aren't authored, they're generated. Extraction and compilation tools produce them as minimal structural representations for simulation and LVS. High-level concepts (block structure, signal intent, mode semantics, architectural boundaries) get intentionally discarded or flattened. The netlist is optimized for tools, not for reasoning about intent.

An LLM reading this netlist has no access to any of it. **Semantic understanding must be constructed, not guessed.**

6.2 DETERMINISTIC ANALYSIS AS A SEMANTIC RECONSTRUCTION

OneCheck performs deterministic electrical analysis on the netlist, but produces semantic information, not more structure.

OneCheck treats the netlist as an electrical system, not a text file. Analysis operates at the systemic level. Behavior emerges from interactions across hierarchy, power domains, and operating conditions. The output: explicit, verifiable semantic objects.

- **Power domains and voltage intent**
  Supply networks, derived rails, operating voltages, and relationships between domains.
- **Propagation paths across hierarchy**
  How signals and currents propagate through instances and hierarchical boundaries.
- **Topology recognition**
  Level shifters, isolation cells, bias networks, protection elements, identified by electrical role, not naming.
- **Electrical constraints derived from the PDK**
  Device-level and topology-dependent operating limits and validity rules.
- **Functional groupings based on rule evaluation**
  Devices and nets participating in the same electrical invariant, independent of schematic hierarchy.
- **Causal relationships between devices and violations**
  Why a violation occurs, not just where, traced through cause-and-effect chains.

This analysis is systemic, not local. Violations aren't independent device failures; they're manifestations of system-level behavior. An Electrical Overstress results from interacting power domains and control logic, not from an isolated transistor attribute.

The output isn't a transformed netlist. It's a set of structured, typed, deterministic semantic objects, each with precise electrical meaning and verification status. This forms the auditable foundation for everything that follows.

6.3 SIMPLIFICATION THROUGH RULE-BASED GROUPING

Semantic reconstruction simplifies through meaning.

Large designs look like millions of independent devices in a netlist. Electrically, they aren't. Devices and nets group according to the electrical rules and behaviors they participate in.

A thousand instances of the same level shifter topology, connected between the same power domains, violating the same electrical constraint, these aren't a thousand separate problems. They're one problem instantiated a thousand times.

Structures responding identically to the same rule collapse into higher-level semantic entities. The grouping isn't based on naming conventions or schematic hierarchy; it's based on electrical role. Two devices in completely different parts of the design, with different instance names, in different hierarchical blocks, can belong to the same semantic group if they participate in the same electrical behavior and respond to the same rule in the same way.

Rule-driven grouping achieves two goals:

- **Complexity reduction**
  Repetitive or equivalent structures collapse into single semantic units.

- **Root cause linking**
  Violations connect directly to their true causes, not their many structural manifestations.

The result isn't a compressed netlist but a small number of stable, meaningful objects representing how the circuit behaves.

6.4 FROM SEMANTIC REPRESENTATION TO EXPLAINABLE ERRORS

Once this semantic layer is established, electrical errors are no longer scattered events across thousands of devices. They become well-defined phenomena with:

- A specific root cause
- A clear scope of impact
- A consistent explanation

Instead of scrolling through thousands of warnings and mentally grouping them, the engineer sees a structured list of actual problems. Each problem has a definition, a cause, and a boundary. The question shifts *from "what are all these violations?"* to *"which of these problems should I fix first?"*

This representation is compact, stable, deterministic, and directly aligned with how designers reason about correctness.

This is what makes reliable explanation, navigation, and interaction possible. The hard work, reconstructing meaning from syntax, is done before AI enters the picture.

7.

CLUSTERIZATION: FROM VIOLATIONS TO ROOT CAUSES

7.1 WHEN ERROR REPORTING STOPS SCALING

Most verification flows report violations as individual error locations: a device, a pin, a net, repeated across hierarchy. This is precise but doesn't scale.

In modern mixed-signal SoCs, a single issue like an invalid domain crossing, a missing isolation structure, or a systematic overstress, can manifest hundreds or thousands of times. **Each occurrence is technically distinct, but they all express the same electrical problem.**

This multiplicity isn't just inconvenient; it's dangerous. A minor issue replicated across hundreds of instances can be harder to fix than a severe isolated problem. It would be harder to correct globally, harder to waive consistently, more likely to interact with firmware, test modes, or future reuse.

Flat reporting creates a familiar pattern. Engineers spend time triaging, sorting, and waiving while the true root cause stays buried under repetitive symptoms.

7.2 ELECTRICAL RULES AS SEMANTIC ANCHORS

OneCheck approaches error analysis from a different angle. Instead of treating electrical rules as simple pass/fail checks, it treats them as semantic anchors that define the nature of the problem being observed.

Each rule expresses not only what must be true electrically, but also the kind of structures and situations to which it applies, and the conditions under which a violation is meaningful. In that sense, a rule already carries an implicit model of the electrical phenomenon it is trying to prevent.

When OneCheck evaluates a design against these rules, it does more than enumerate violations. It identifies which parts of the circuit participate in the same underlying electrical situation as defined by the rule. This observation is the starting point for clusterization.

7.3 WHAT DO WE CALL A CLUSTER?

A cluster is OneCheck's way of turning many local observations into a single, meaningful design issue.

A cluster groups violations that are different manifestations of the same root cause. They may appear on different devices, at different hierarchy levels, across repeated instances, but they originate from the same underlying mistake or missing intent.

From the designer's perspective, this reflects how problems are actually fixed. They are not fixed by correcting hundreds of individual error sites, but by addressing the single architectural decision that caused them.

Designers don't fix "842 violations". They fix one problem that happens to appear 842 times.

A cluster has three defining elements:

- **Rule context**: which electrical rule is violated
- **Causal mechanism**: why the violation occurs
- **Design origin**: where it originates in the architecture

Everything in a cluster can be explained consistently and fixed with the same reasoning.

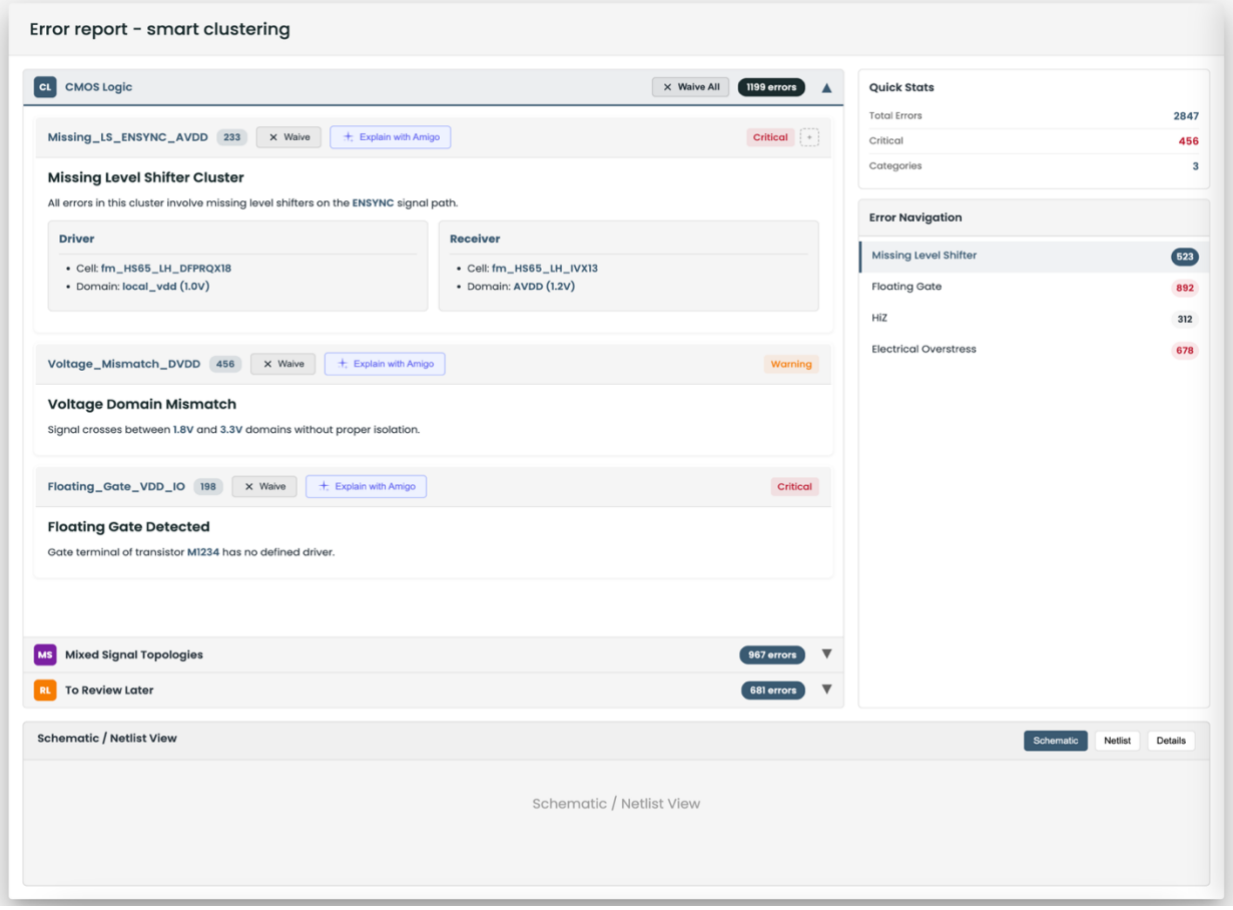

Figure 3: OneCheck smart clustering interface—violations grouped by root cause

7.4 HOW CLUSTERIZATION IS CONSTRUCTED

Clusters aren't produced by superficial pattern matching or post-processing error logs. They're derived directly from the semantic representation built during deterministic analysis.

The semantic representation encodes power domains, propagation paths, device roles, constraints, and cross-hierarchy relationships. Using this, OneCheck traces each violation back to its electrical origin. Violations sharing the same origin (same signal path, same domain boundary, same control dependency) cluster together.

Only violations related by causality cluster together. This makes clustering deterministic. You got for the same design and the same rules the same clusters. Results don't depend on reporting order or incidental structure.

During semantic reconstruction, the circuit is analyzed from multiple perspectives: causality, power domain boundaries and signal roles, extracting the functional properties that determine how devices should group.

EXAMPLE: CLUSTERING DEPENDS ON ANALYSIS PERSPECTIVE

Consider four inverters in series, alternating between power domains VDD1 and VDD2.

How should these devices cluster? It depends on what property you're analyzing:

By causality: If both VDD1 and VDD2 are ON, all four inverters share a common signal path through the "inCell" net. From a causality perspective, they form one cluster and they all depend on the same input.

By power domain crossing: Three of the four inverters have their gate in one domain and their drain/source in another. The alternating domains create two distinct crossing patterns: VDD1→VDD2 and VDD2→VDD1. These form two separate clusters. Depending on power scenarios, one cluster may be irrelevant and filtered out.

Grouping is hierarchical. Devices cluster by one property, then sub-cluster by another, forming a tree. At the leaves you get the devices that are functionally equivalent for a specific set of properties. This is how 12,000 raw violations might reduce to 50 distinct root causes.

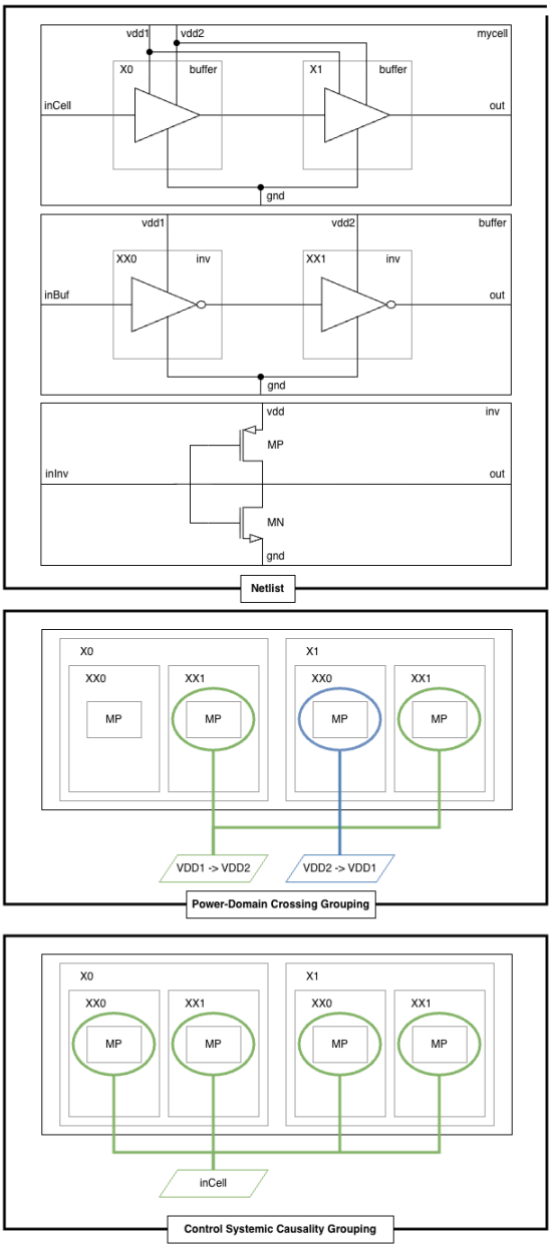

Figure 4: The same circuit clusters differently depending on analysis perspective—causality grouping vs. power domain crossing grouping

7.5 WHY CLUSTERIZATION MATTERS AND WHY IT ENABLES AI

Traditional ERC lists violations individually. Thousands of entries that engineers must mentally group. Clusterization does this automatically.

With clusterization, that list transforms. Instead of thousands of individual violations, designers see a small number of well-defined problems, each with a distinct root cause. A report that once showed 12,000 violations might now show 50 clusters. Each cluster has a name, a cause, a scope, and an impact assessment. Cognitive load drops immediately. Prioritization becomes possible by fixing the cluster affecting 3,000 instances before the one affecting 12. Effort shifts from triage to actual correction, at the architectural level.

This brings organizational consistency. Waivers attach to root causes, not duplicated across instances. When a team decides that a particular violation is acceptable under specific conditions, that waiver applies to the entire cluster, not to 2,847 individual device entries that need separate sign-off.

Explanations and design guidelines work the same way. Document the fix for one cluster, and it covers every manifestation. Recurring issues become identifiable engineering themes that persist across projects. No more scattered error IDs that no one remembers from one tapeout to the next.

For AI, clusterization creates a safe interface. Without it, an LLM faces thousands of isolated violations, each a small fragment of netlist with minimal context. The model would have to guess which violations are related, infer root causes from syntax, and produce consistent explanations across instances it doesn't know are connected. This is exactly where LLMs fail. Too little structure, too much ambiguity, no way to verify the output. Clusterization eliminates this problem. Instead of thousands of fragments, OneCheck presents one electrical problem at a time, compact and semantically coherent, with verified scope and precise root cause.

Each cluster encapsulates exactly what's needed to explain the problem: what's wrong, why it's wrong, where it originates, how it manifests across the design. This is the natural boundary between deterministic analysis and AI interaction.

When Amigo receives a cluster, it receives a complete problem definition, not a puzzle to solve. The LLM's job becomes synthesis and explanation, translating structured electrical facts into clear language, without inference and guessing.

Explanations produced this way aren't speculative. They're anchored in verified electrical reality. Ask about cluster #7 today, get an answer grounded in its causal chain. Ask tomorrow, get the same answer. This reproducibility is what makes AI assistance trustworthy for sign-off workflows.

The intelligence comes from the model; the correctness comes from the structure underneath.

8.

SYSTEM CONDITIONAL ANALYSIS: EXPLORE CIRCUIT STATES FOR ANALOG CAUSALITY

8.1 THE LIMITS OF PURELY STATIC ANALYSIS

Static checks are conservative. They flag every path that could violate a rule, regardless of whether that path is ever active.

Consider a level shifter violation detected between two power domains, VDD_1V0 and VDD_1V2. Static analysis sees the structural connection and reports the error. But whether this violation matters depends on questions structure alone cannot answer:

- Are both domains ever active simultaneously?
- What is the actual voltage on the driving node when the receiver is powered?
- Does the control logic permit the problematic state?

In real designs, many structural violations are functionally unreachable. Power sequencing, enable logic, and mutual exclusion constraints eliminate entire classes of scenarios. Without accounting for these, verification produces excessive false positives, consuming engineering time and reducing trust in results.

The solution is not to abandon static analysis, but to extend it with conditional reasoning that determines when a structural violation becomes an electrical reality.

8.2 THE CORE PROBLEM: CONDITIONAL REACHABILITY

The question is conditional reachability. Under what conditions does a structural violation become an electrical reality?

This is a reachability problem, but the state space is too large to enumerate.

SysCon solves this through reverse conditional propagation. Instead of searching forward through all possible states, it works backward from the violation, reconstructing only the conditions necessary for that specific error to manifest

.

PRACTICAL EXAMPLE C: **THE ZOOM LEVEL PARADOX**

This example shows how the same circuit reaches different conclusions at different observation scopes.

LEVEL 1: DEVICE VIEW (INSIDE THE IP BLOCK)

You examine signal CTRL and see:

- Driver: DFF output Q in block CORE_LOGIC, powered by VDD_1V0
- Receiver: Inverter input A in block IO_BUFFER, powered by VDD_1V8
- Level shifter: None

Conclusion: **ERROR** – Missing level shifter on domain crossing.

LEVEL 2: BLOCK VIEW (IP BOUNDARY)

You zoom out and discover:

- The inverter INV_X1 is inside an isolation cell ISO_CLAMP
- When ISO_EN = 1, the output is clamped to VSS regardless of the input voltage
- The isolation cell handles the domain crossing

Conclusion: **OK** – The isolation cell makes the level shifter unnecessary.

LEVEL 3: SYSTEM VIEW (SOC INTEGRATION)

You zoom out further to examine power modes:

- In SLEEP_MODE:
  - VDD_1V0 (core domain) is powered down
  - VDD_1V8 (I/O domain) remains on
  - ISO_EN is generated by logic in the core domain... which is now unpowered
  - Therefore ISO_EN floats

Conclusion: **ERROR RETURNS** – The isolation cell exists but doesn't activate because its control signal is unpowered.

WHAT IT DEMONSTRATES

The same circuit reaches three different conclusions depending on observation scope:

1. Device level: Error (missing level shifter)
2. Block level: No error (isolation handles it)
3. System level: Error (isolation control unpowered)

A local check, a block-level check, and a system-level check reach contradictory conclusions. Only the system-level analysis captures the real behavior.
This is exactly what System Conditional Analysis solves.

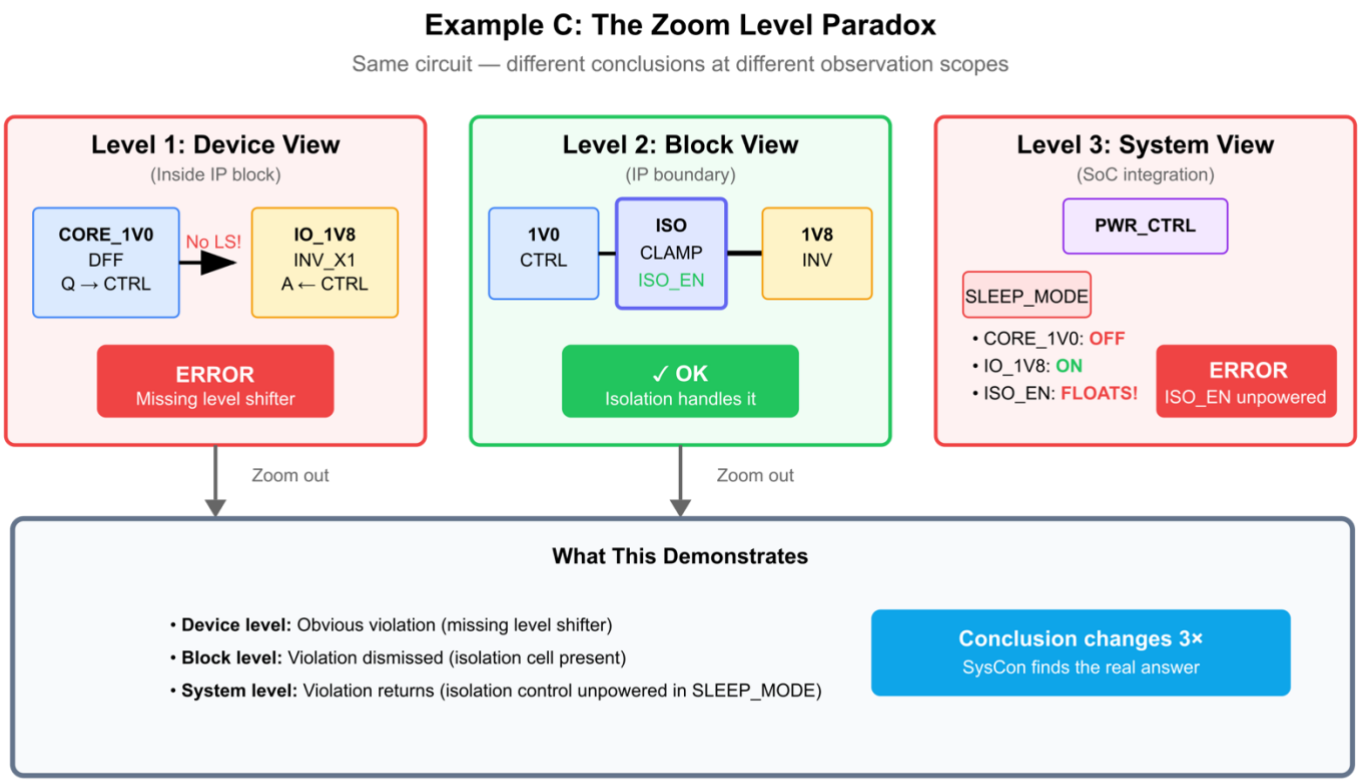

Figure 5: The Zoom Level Paradox—same circuit, different conclusions at different scopes

8.3 Reverse Analysis from the Point of Interest

SysCon operates by performing a reverse analysis starting from a point of interest, typically an electrical violation or a suspected error location.

It performs backward symbolic propagation from the violation point. It identifies controlling elements (switches, regulators, bias dependencies) and infers the minimal conditions required for the violation to manifest. No topology encoding required.

8.4 CONDITIONAL REASONING ACROSS ANALOG AND DIGITAL DOMAINS

SysCon reasons across both analog and digital control structures. It operates in this mixed context.

The analysis considers:

- Digital control states (enable signals, configuration registers)
- Analog conditions (bias presence, voltage levels, threshold behavior)
- Hybrid interactions where digital signals gate analog paths

8.5 EXTRAPOLATING ERRORS TO THE SYSTEM LEVEL

Beyond verification, SysCon enables extrapolation. Once the causal chain of a violation is identified, the analysis can express the problem in terms of higher-level signals or system interfaces.

Errors can be described not just at the device level, but in terms of:

- Control signals responsible for enabling the path
- Power sequencing assumptions
- System-level pins or operating modes
- Biasing states and current references state

This capability is essential for communicating issues effectively, especially in large teams where the designer fixing the problem may not be the one who implemented the low-level circuitry.

8.6 CONSTRAINT INJECTION FOR DESIGN-SPECIFIC ASSUMPTIONS

Designs carry implicit assumptions that aren't in the netlist:

- "VDD_CORE and VDD_IO are never both off while RETENTION_EN is high"
- "The PLL must be locked before ADC_CLK_EN is asserted"
- "During scan test, all analog blocks are isolated"

SysCon accepts these as explicit constraints that restrict the analysis space. Designers can inject assumptions at multiple levels:

Constraint Type	Example	Effect
Power sequencing	`AVDD enables before DVDD`	Eliminates transient cross-domain states
Mutual exclusion	`NOT(MODE_A AND MODE_B)`	Prunes impossible configurations
Functional assumptions	`BIAS_VALID implies IBIAS > 1µA`	Refines analog intervals
Test mode isolation	`SCAN_MODE implies ANALOG_ISO = 1`	Separates functional and test analysis

Injecting accurate constraints reduces false positives without globally disabling checks.

The constraint applies only where relevant, preserving coverage for scenarios that remain reachable.

8.7 THE TRANSFORMATION

Without conditional validation, AI can't distinguish real violations from structural artifacts. An LLM presented with 1,000 raw violations cannot know which matter without operating context.

When Amigo receives a SysCon-validated violation, it operates on verified facts:

- The violation WILL occur under condition X
- The root cause is signal Y driving across domain Z
- The impact propagates through path P

The AI adds value through natural language synthesis and design guidance; the underlying correctness comes from deterministic analysis.

Before SysCon	After SysCon
1,000 structural violations	127 reachable violations
Unknown relevance	Each tagged with enabling condition
No causal chain	Clear path from control signals to error
Cannot prioritize	Ranked by scenario likelihood

PRACTICAL EXAMPLE D: **CONNECTIVITY ≠ REACHABILITY**

WHAT STATIC ERC SEES

Static analysis examines the netlist and finds:

- A conductive path from `VDD_CORE` to `VDD_IO`
- The path passes through switches and logic in both domains
- No isolation cell exists on this path

ERC flags **847 violations across all instances of this pattern.**

WHAT SYSTEM ANALYSIS REVEALS

SysCon traces the control logic and discovers:

- `VDD_CORE` is gated by signal `EN_CORE` from the power controller
- `VDD_IO` is gated by signal `EN_IO` from the power controller
- The power controller `FSM` enforces a constraint:

```
VALID_POWER_STATES:
  - ACTIVE:    EN_CORE = 1, EN_IO = 1  //  Not allowed by FSM
  - CORE_ONLY: EN_CORE = 1, EN_IO = 0  //  Valid
  - IO_ONLY:   EN_CORE = 0, EN_IO = 1  //  Valid
  - SLEEP:     EN_CORE = 0, EN_IO = 0  //  Valid
Invariant: NOT(EN_CORE AND EN_IO) in all reachable states
```

The path exists structurally, but the two domains are never powered simultaneously in any valid operating state. The path is never electrically active.

WHAT IT DEMONSTRATES

Analysis Type	Sees	Concludes
Static ERC	Path exists	847 violations
SysCon	Path exists but never active	False positive

The structural violation is real. The electrical violation is impossible.

Connectivity ≠ Reachability

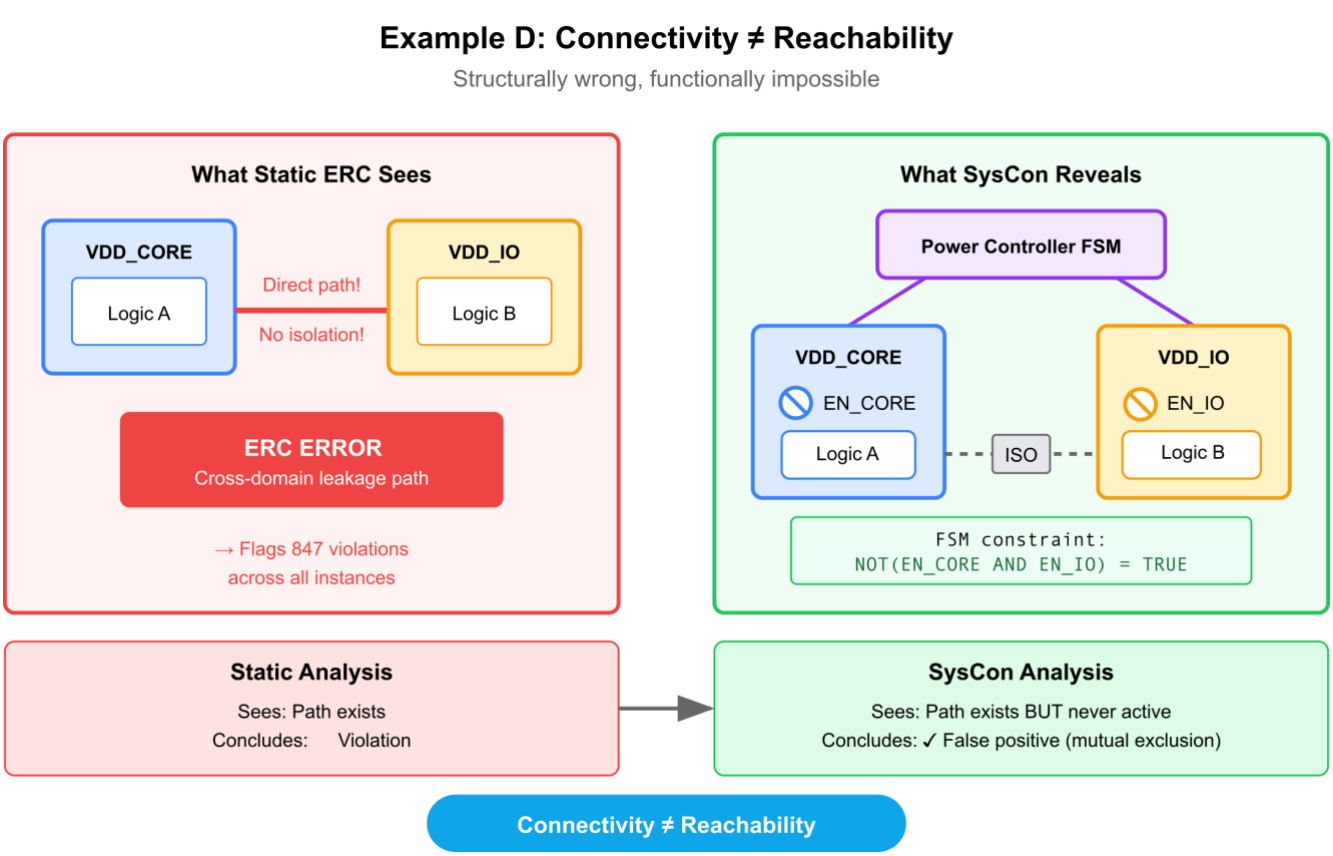

Figure 6: Connectivity is not Reachability—structurally wrong, functionally impossible

9.

AMIGO: AI ON TOP OF VERIFIED ANALYSIS

9.1 WHERE AI FITS

OneCheck builds a semantic model. Clusterization groups violations by root cause. SysCon filters out the unreachable ones. What remains is verified, causally grounded problems ready for human review.

Within this architecture, AI is introduced after analysis, not in place of it.

Amigo doesn't detect errors—OneCheck does. Amigo takes the verified semantic model and makes it usable for explanations, navigation, and fix suggestions.

Deterministic analysis handles correctness. AI handles interaction. This separation is intentional, and it's what keeps the system predictable and auditable.

9.2 THE ARCHITECTURE

Amigo connects to OneCheck through MCP (Model Context Protocol), a structured interface that exposes the semantic model to an LLM. An MCP is as a controlled window into the analysis results: the LLM can query the model, request specific data about violations, ask for propagation paths or power scenarios, and get structured responses back. It doesn't replace any analysis; it's a query layer that sits on top of verified results.

The interface is bidirectional. Amigo can request information ("show me the causal chain for this violation"), and the semantic model returns typed, structured data. Not free-form text.

The stack:

- **OneCheck** → semantic model from netlist
- **SysCon** → filters to reachable violations
- **Clusterization** → groups by root cause
- **Amigo** → LLM interface to query and explain results

Each layer adds structure. By the time Amigo sees a violation, it's verified, scoped, and causally grounded.

Amigo isn't a report generator. It's interactive. An engineer asks a question, Amigo queries the semantic model, gets structured data back, and synthesizes an answer. The engineer asks a follow-up; Amigo digs deeper.

The key property is that every response traces back to deterministic analysis. The LLM synthesizes and explains, but the facts come from OneCheck.

The model can only see what's in the semantic model. It can't invent violations or operating conditions that weren't analyzed; though it may still produce imprecise explanations that need review.

This semantic model is the single source of truth, and Amigo can only see what's in it.

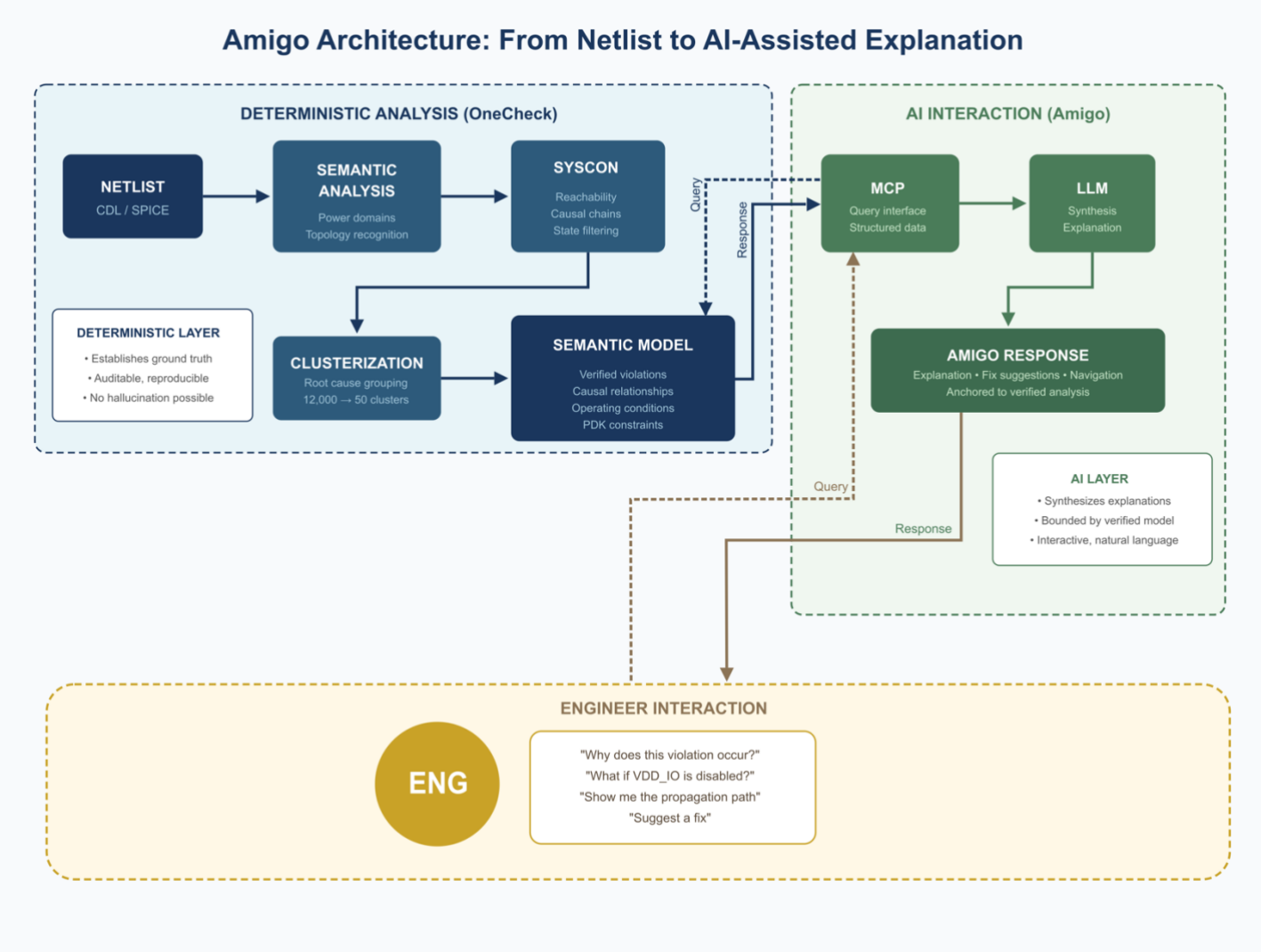

Figure 7: End-to-end architecture from netlist to AI-assisted explanation. OneCheck performs semantic analysis, SysCon filters to reachable violations, and clusterization groups by root cause—producing a verified semantic model. Amigo connects to this model through MCP, allowing engineers to ask questions and receive grounded answers. The separation between deterministic analysis (blue) and AI interaction (green) ensures that all explanations trace back to verified results.

9.3 WHY SPEED MATTERS

OneCheck is fast. Analysis runs in minutes, not hours. Fast enough that results can be queried interactively rather than consumed as a static report.

Once OneCheck builds the semantic graph (connectivity, power intent, topology, causality) it can be queried repeatedly without re-running analysis. The cost of understanding the circuit is paid once; subsequent interactions operate on the already-built model.

This enables a different usage pattern. From a cluster, engineers can ask:

- *"How does this violation behave under operating condition X?"*
- *"Which control signal enables this path?"*
- *"What if this supply is disabled?"*
- *"Is this topology reused elsewhere?"*

Through MCP, Amigo can:

- Navigate the semantic model
- Traverse hierarchy
- Follow propagation paths
- Request focused evaluations of specific states

These queries are incremental. No need to reprocess the netlist or recompute rules, the model is already there.

The loop is simple. Analysis builds the model once, then Amigo asks questions against it. Follow-ups don't require reprocessing.

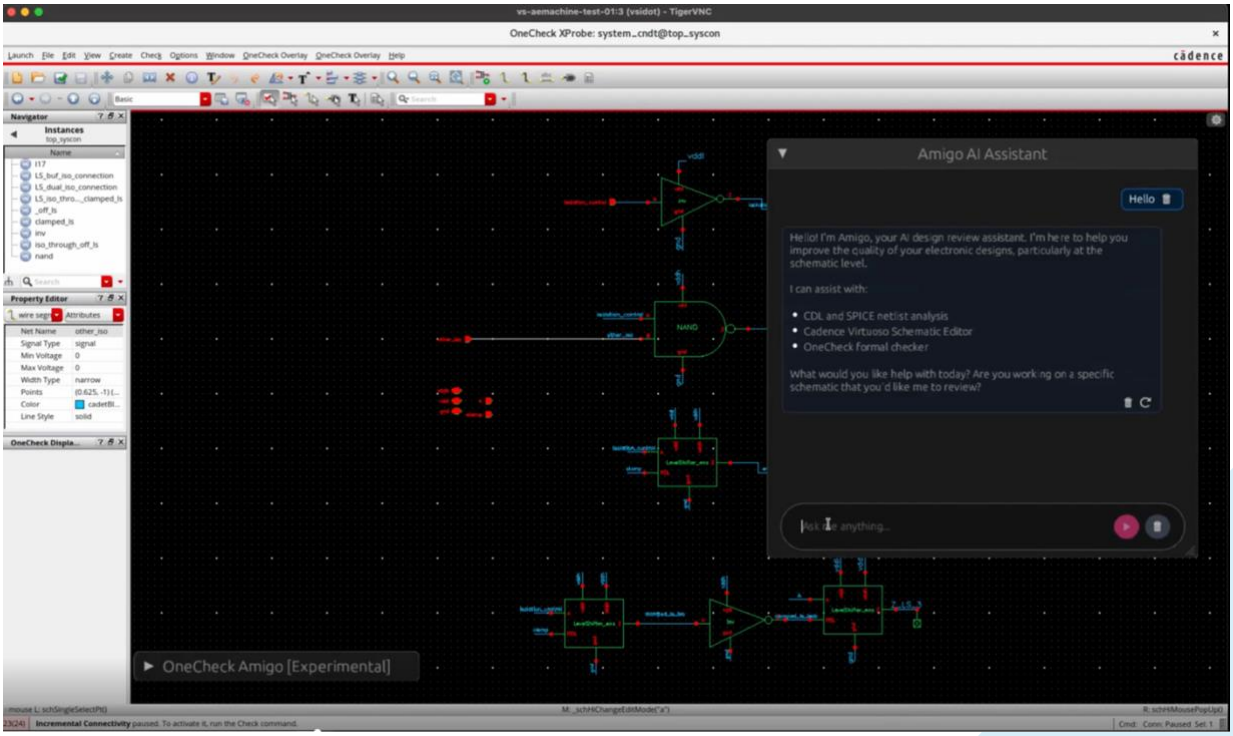

Figure 8: Amigo integrated in Cadence Virtuoso—AI assistant panel alongside schematic view

9.4 WHAT CONTEXT THE MODEL RECEIVES

Each Amigo interaction is anchored to a specific cluster. For that cluster, the LLM receives structured context:

- **Rule:**
  which electrical rule was violated, and its definition
- **Cluster composition**:
  which devices and nets are involved, why they're grouped
- **Netlist fragment:**
  the relevant CDL/SPICE, not the whole design
- **Power scenarios:**
  which supplies are active, voltage levels
- **Propagation paths:**
  how signals flow through the violation
- **PDK constraints:**
  device ratings, topology rules
- **SysCon conditions**:
  under what states this violation is reachable

This context is curated to provide minimum sufficient information to explain the issue. The LLM doesn't see millions of devices or the full design hierarchy. It sees the relevant fragment, annotated with semantic information, typically a few hundred lines of context per cluster.

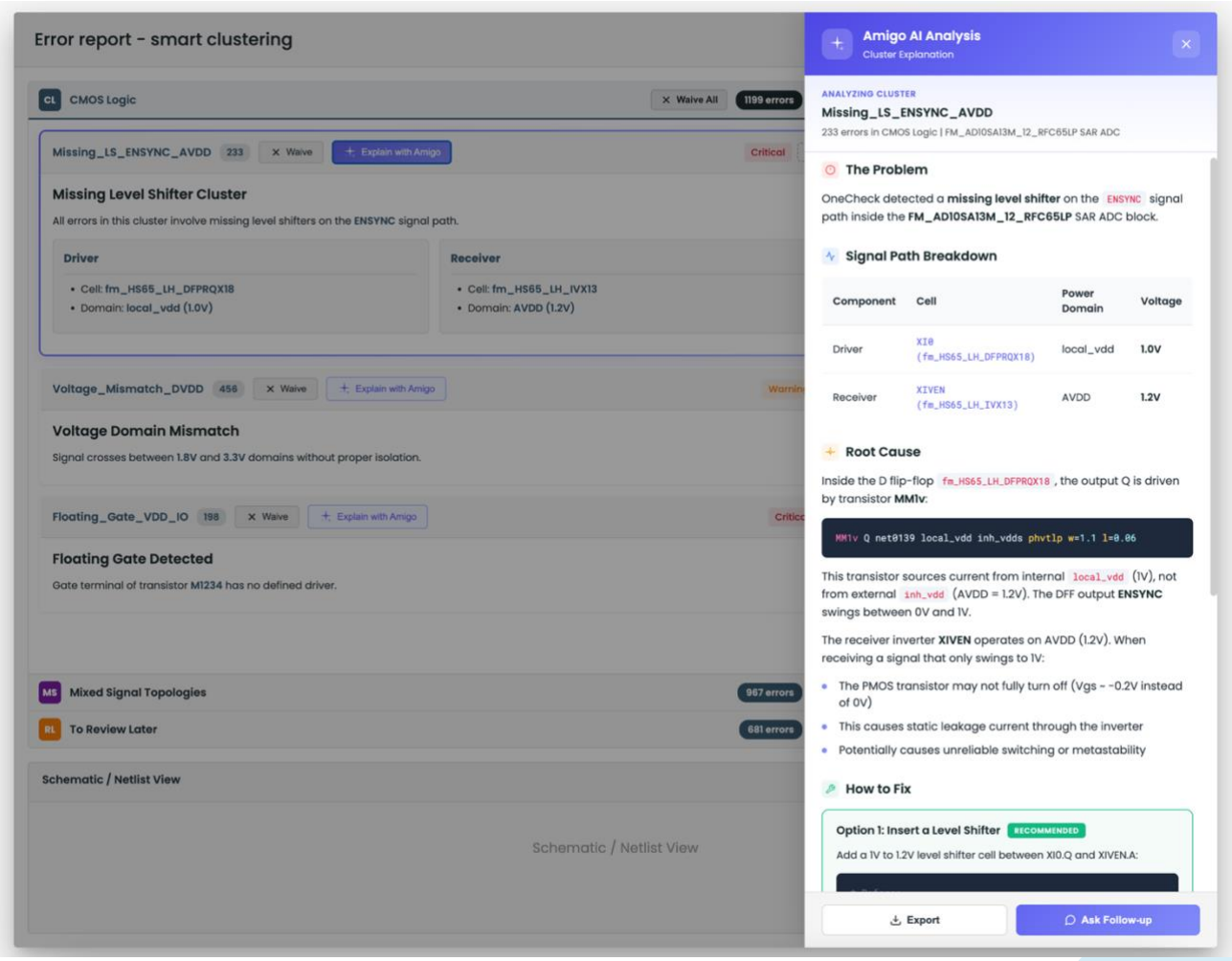

Figure 9: Amigo cluster interaction—structured context displayed alongside AI analysis

9.5 WHAT AMIGO DOES

Given this context, Amigo supports several classes of interaction that go beyond static report generation.

EXPLAIN THE ERROR IN NATURAL LANGUAGE

Amigo explains what's wrong, why, and when it matters. Example:

```
"This cluster contains 233 missing level
shifter violations on the ENSYNC signal
path.

The driver (DFF in fm_HS65_LH_DFPRQX18)
operates at 1.0V in the local_vdd domain.
The receiver (inverter in
fm_HS65_LH_IVX13) operates at 1.2V in the
AVDD domain.

When ENSYNC swings 0→1V, the receiver sees
insufficient Vgs, potentially failing to
switch fully.

This causes static leakage and risks
metastability."
```

This explanation is consistent across all 233 instances because they share the same causal structure.

SUGGEST FIXES

Amigo proposes fixes in context. For example, regarding the previous ENSYNC violation:

```
"Option 1: Insert a 1V-to-1.2V level
shifter between the DFF output and the
inverter input. This preserves signal
timing and eliminates the domain crossing
issue.

Option 2: Move the DFF to the AVDD domain.
This eliminates the crossing but may
impact power consumption—the 1.2V domain
draws more current and the DFF would now
be always-on."
```

The suggestions reference actual cells, constrains and domain boundaries from the design, not generic advice.

NAVIGATE THE DESIGN

Engineers can ask *"Show me the propagation path from the clock source to this violation"* or *"Where else is this topology used?"*

Amigo queries the semantic model and returns structured answers, no manual netlist tracing.

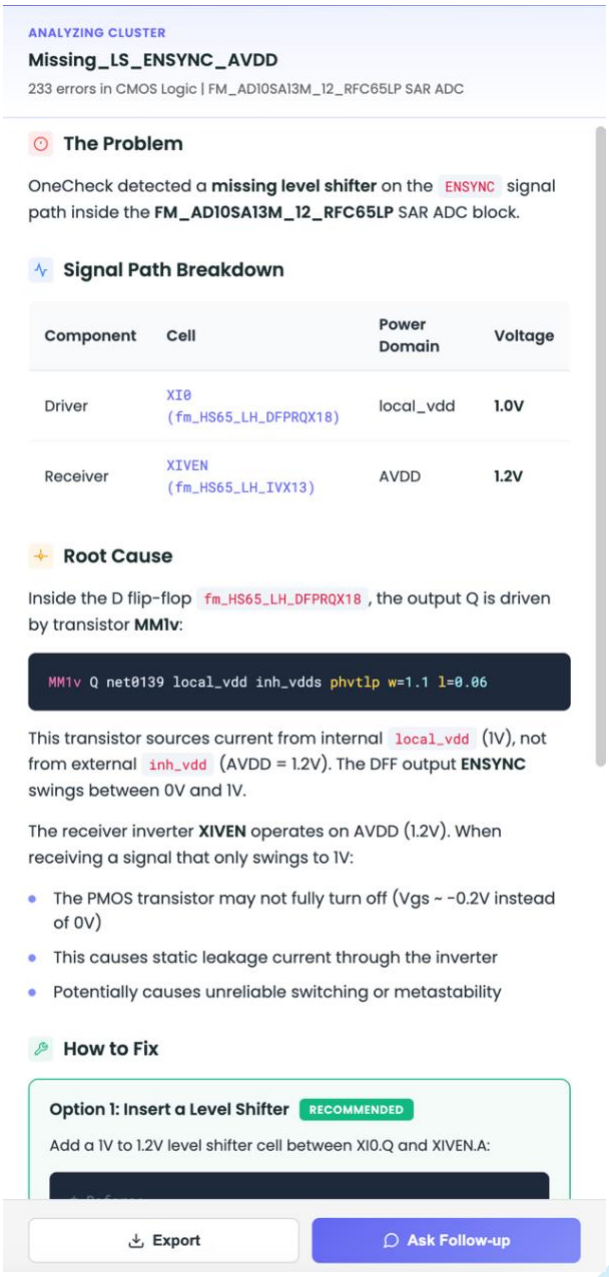

Figure 10: Amigo cluster analysis—problem breakdown, root cause, and fix option

ASK FOLLOW-UP QUESTIONS

Because Amigo queries the semantic model, it can answer:

- ***"Is this violation present in all operating modes?"***
- ***"What signal controls this path?"***
- ***"What if VDD_IO is disabled?"***

Answers stay anchored to the verified model. No drift. No hallucination.

9.6 BOUNDARIES

Amigo operates within the semantic model. It can't detect errors that OneCheck missed, override analysis results, invent assumptions outside the constraint set, or explore states that SysCon marked unreachable.

These constraints exist by design. The LLM explains and synthesizes. The deterministic engine establishes ground truth. If Amigo could override analysis results, you'd lose the auditability that makes the system trustworthy.

Amigo can also produce imprecise or misleading explanations of real violations. The facts come from OneCheck and the synthesis comes from the LLM. Engineering review remains necessary, particularly for novel topologies or unusual operating conditions.

9.7 THE POINT

ERC tools produce violation lists. Amigo turns those lists into something usable. Explanations you can read, fixes you can evaluate, follow-up questions you can ask. The goal isn't to replace engineering judgment. **The goal is to stop wasting engineering time on triage**.

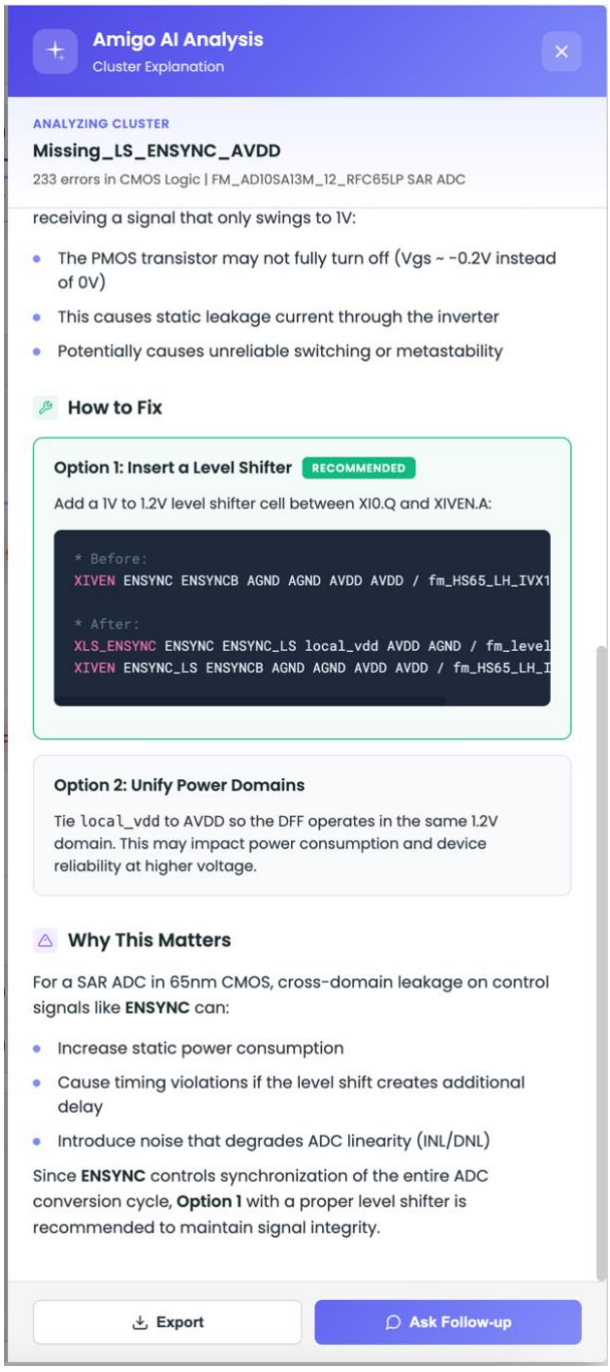

Figure 11: Amigo fix suggestions with before/after netlist fragments

10 LIMITATIONS AND FUTURE DIRECTIONS

10.1 CURRENT LIMITATIONS

This architecture makes trade-offs. Understanding the boundaries helps set appropriate expectations.

Rule coverage determines detection scope. OneCheck detects violations covered by its rule set and PDK integration. Novel failure modes outside this scope won't be flagged and there's no warning that something was missed. This is inherent to rule-based verification: you find what you look for. Expanding coverage requires adding rules, not tuning parameters.

SysCon is only as good as its constraints. If operating assumptions are missing or wrong, SysCon may incorrectly filter out real violations.

Example: If the constraint set doesn't capture that TEST_MODE can be active during normal operation, violations reachable only in TEST_MODE get filtered as 'unreachable', but they're real.

A missing constraint silently removes coverage.

Amigo can't explain what OneCheck didn't find. The AI operates on the semantic model. If something isn't in the model (because the rule doesn't exist or SysCon filtered it) Amigo has nothing to say about it. No hallucinated warnings, but also no coverage beyond the analysis scope.

Current scope is transistor-level electrical verification. The architecture could extend to timing, layout, or reliability verification, but each domain requires its own rule set, PDK integration, and semantic model. This isn't a configuration change, it's new development.

No quantitative benchmarks in this paper. We describe architecture, not performance. Runtime scaling, memory usage, and accuracy metrics against known violation sets are future work.

10.2 FUTURE DIRECTIONS

Broader rule coverage. New circuit techniques, device types, and reliability concerns require new rules. This is continuous work driven by real designs hitting edge cases.

Earlier integration in the design cycle. Tighter coupling with schematic editors and layout tools would let Amigo assist during design, not just during verification. Catching issues before extraction saves more time than explaining them after.

Learning from design history. Patterns across projects reveal recurring issues. A system that tracks "this topology caused problems in three previous designs" could suggest preventive measures before violations appear. This requires careful handling of proprietary design data.

Extension to other verification domains. Timing closure, electromigration or thermal analysis, the same architecture applies. Deterministic analysis builds the semantic model while AI explains and interacts. Each domain needs its own rules and models, but the pattern transfers.

11.

Conclusion

Can you feed a netlist to an LLM and get reliable verification results?

The answer is no.

Not because LLMs aren't capable, but because netlists don't contain the information needed for correct reasoning. Design intent, power semantics, operating modes, causal relationships: none of this survives extraction.

OneCheck builds a verified model from the netlist. SysCon filters to reachable states. Clusterization groups violations by root cause. Amigo explains, synthesizes, and assists. The deterministic engine establishes what's true.

When an engineer asks "why does this violation occur?", the answer traces through causal chains, power scenarios, and PDK constraints. The explanation is reproducible. Another engineer asking tomorrow gets the same answer.

Where this leads?

AI won't replace analog verification. It will change how engineers interact with results. The shift from "decode this error report" to "explain this and suggest what to do" is significant. But it only works if the underlying analysis is trustworthy. Get that wrong, and AI becomes a fluent generator of unverifiable explanations.

Context isn't optional. It's the foundation.

Aniah
Zero Electrical Error on 1st Silicon Spin
www.aniah.fr