Bi-directional Data Flow in VLSI design using Ontology and KG

Ilhami Torunoglu – Siemens, 2026

SIEMENS

Agenda

Problem statement

Ontology and knowledge graph to rescue

An illustrative example

KGs impact on common issues

Conclusion

Problem definition

- **Weak pre-layout vs. post-layout correlation**:

  wire RC, placement/congestion, and IR/EM cause late surprises.

- **Siloed data & semantics**:

  RTL, SDC, DEF/LEF, DRC/DFM reports, fab/test data lack a common language.

- **Lengthy ECO loops:**

  root-cause tracing across RTL⇄synthesis⇄P&R⇄sign-off is manual and error-prone.

- **DFM & yield learning gap**:

  manufacturing hotspots and defect patterns rarely feed back into early design.

- **Test blind spots:**

  limited linkage between at-risk layout patterns and targeted test content.

1 Footnote Arial Regular, 9 pt, bottom aligned

An example (pretty realistic) and how Ontology and KG can help

During late sign-off, a critical path fails timing due to unexpected coupling on M4 near a macro corner.

The bidirectional flow pushes this feedback upstream;

The **KG** traces the path→nets→layer region and finds a recurring **DFM hotspot pattern**.

The **ontology** maps that pattern to specific spacing/density rules and links to past wafer fail bins.

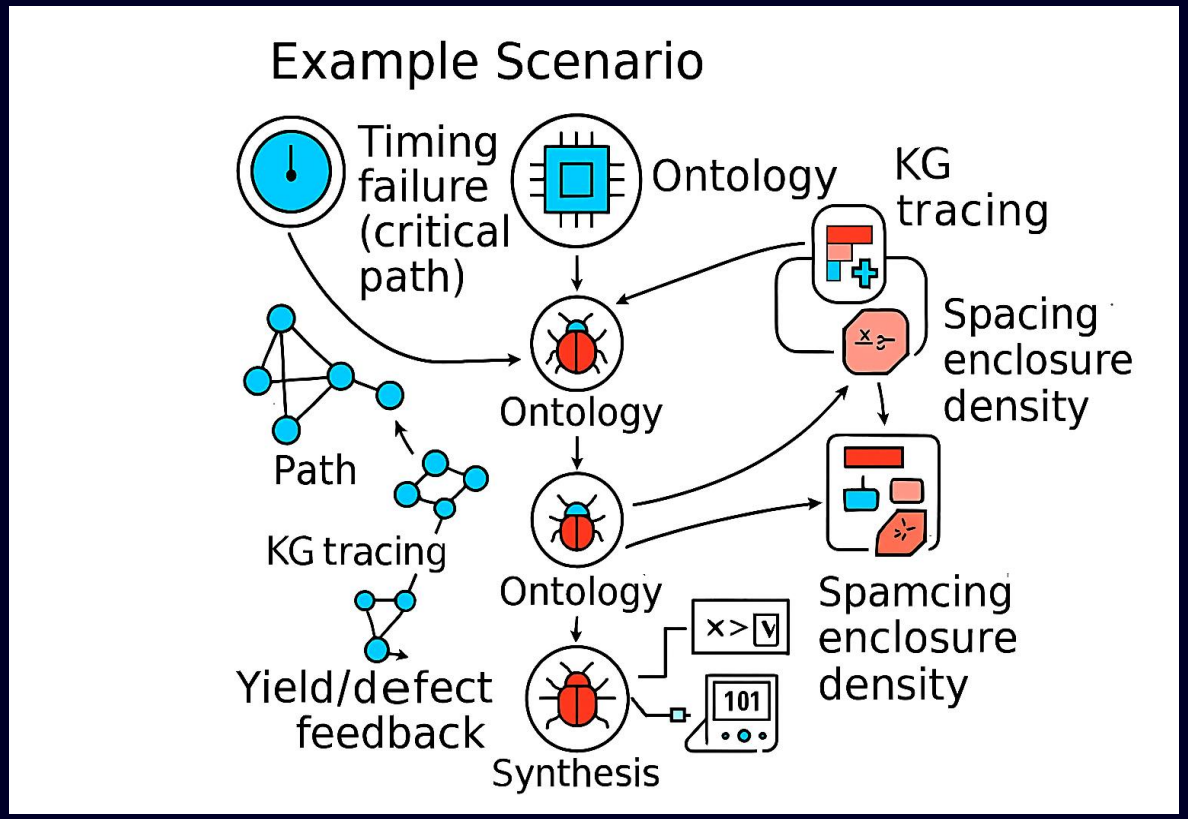
The flow automatically:

(1) applies targeted buffer insertion and spacing constraints in synthesis/P&R,

(2) updates DFM guidance for that region, and

(3) adds a focused test for nets using the pattern.

Result: the next iteration meets timing with fewer ECOs, and risk is reduced by addressing the manufacturability root cause early.



Prior work to solve some/all of these problem

- Graph-based ML improves placement/timing
  - but lacks semantic ontology layer for cross-tool reasoning.
- Manufacturing ontologies (OntoCommons/NIST) unify factory semantics
  - but don’t map cleanly to PDK rules.
- Yield/defect KGs model wafers/defects
  - but rarely connect back to design optimization.
- Equipment/supply-chain KGs offer decision automation
  - but sit outside RTL→GDS loops.
- EDA-Schema provides a cross-stage graph data model
  - but lacks a full domain ontology.

Gaps remain in unifying semantics across design→fab→test flows.

Successes vs. gaps current systems

Successes:

(1) graphs accelerate core EDA tasks;

(2) manufacturing ontologies prove cross-domain interoperability;

(3) KGs help yield/ops decisions.

Gaps:

(A) absence of a widely adopted ontology that unifies RTL/constraints/PDK/rules/DFM/test;

(B) limited KG-driven bidirectional feedback into synthesis/physical optimization;

(C) scarce open benchmarks that span design↔fab↔test;

(D) operationalization challenges (governance, performance, versioning) for silicon

Despite recent progress, **there is still no EDA-wide official ontology**

Because

- EDA flows differ drastically between vendors.
- PDKs are proprietary, inconsistent, and poorly standardized.
- Yield/test data is guarded by foundries.
- No unified representation spans **RTL** → **synthesis** → **P&R** → **DFM** → **fab** → **test**.

However, the building blocks now exist:

- **EDA-schema** → structure for circuits.
- **Sem4EDA** → ontology reasoning.
- **SemicONTO** → technology/material ontology.
- **Supply-chain KGs** → integration with fab + operations.

A **full EDA–DFM–Manufacturing Knowledge Graph** is now feasible for the first time (2025–2026).

How does Bidirectional flow + KG + Ontology helps

- **Bidirectional flow ensures constraints/feedback travel **both ways**
  (front-end ⇄ back-end)

- **Ontology provides a formal schema**

  E.g. (layers, vias, rules, constraints), enforcing consistent interpretation across tools.

- **Knowledge Graph links **entities & relationships**

  (cells, nets, paths, rules, process steps, yield/test) for semantic queries and ML.

- **Closed-loop DFM:**

  fab/test insights (hotspots, fail bins) update rules/constraints, improving future iterations.

- **Faster, data-driven ECOs:**

  graph queries trace violations to causes and propose fixes (buffering, resizing, floorplan tweaks).

Ontology vs Knowledge Graph
A modern approach

An ontology provides the schema and semantics, while a knowledge graph is the instantiation with actual data

Ontology

- Abstract model of concepts and relationships
- Defines what can exist and how things relate
- Contains rules, constraints, and axioms
- Domain-specific vocabulary
- Example: OWL, RDFS specifications

Knowledge Graph

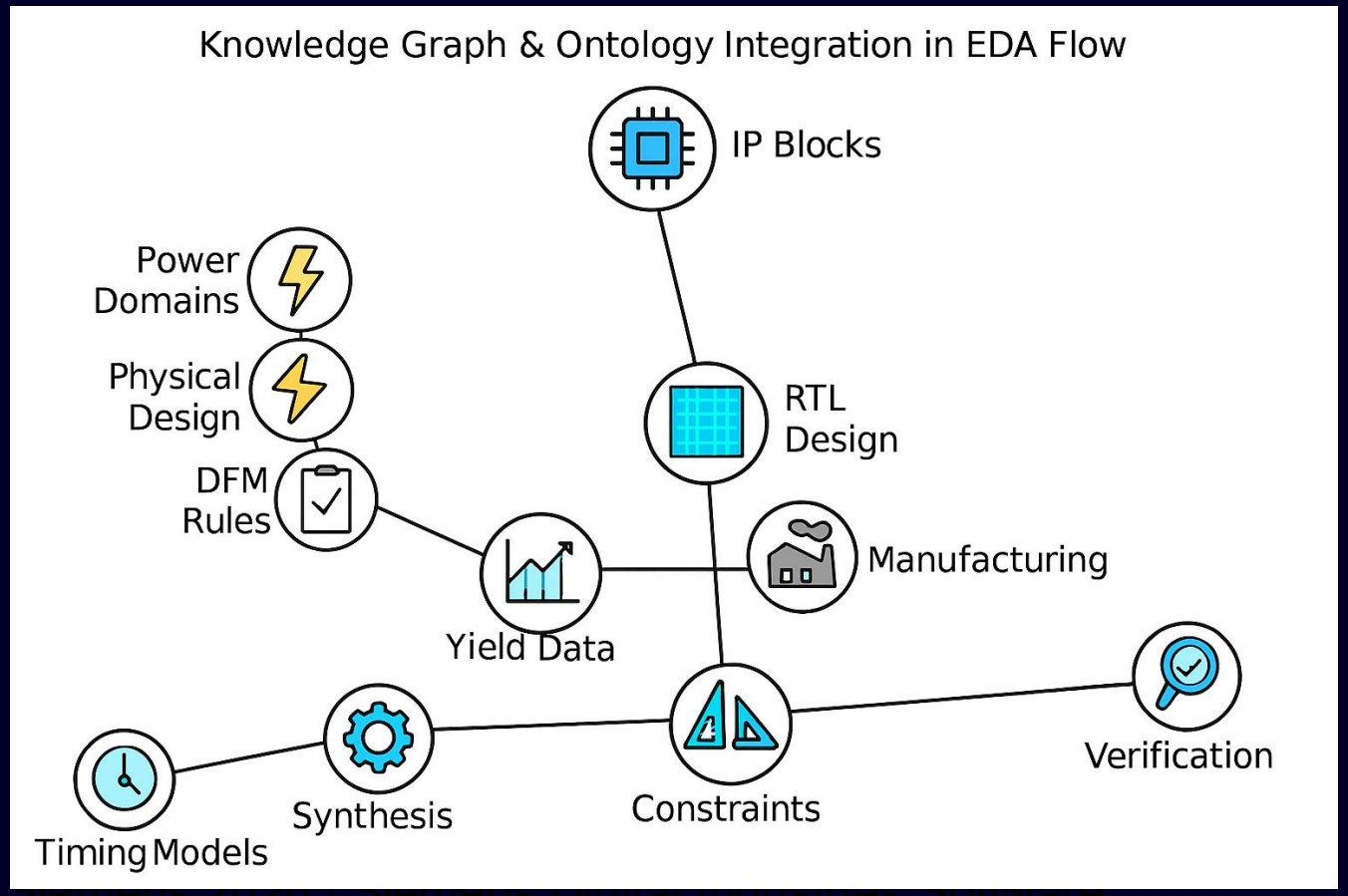
- Concrete instance of data following the ontology
- Contains actual entities and their connections
- Can be queried, traversed, and analyzed
- Evolves with new data
- Example: Google KG, Wikidata

Analogy: An ontology is like a database schema that defines tables and relationships, while a knowledge graph is like the database filled with actual records.

Knowledge Graph & Ontology for EDA

Knowledge graph connects design features, constraints, timing/power models, DFM rules, and manufacturing yield data.

Ontology provides the schema/semantics so data from different tools is interpreted consistently across the flow.



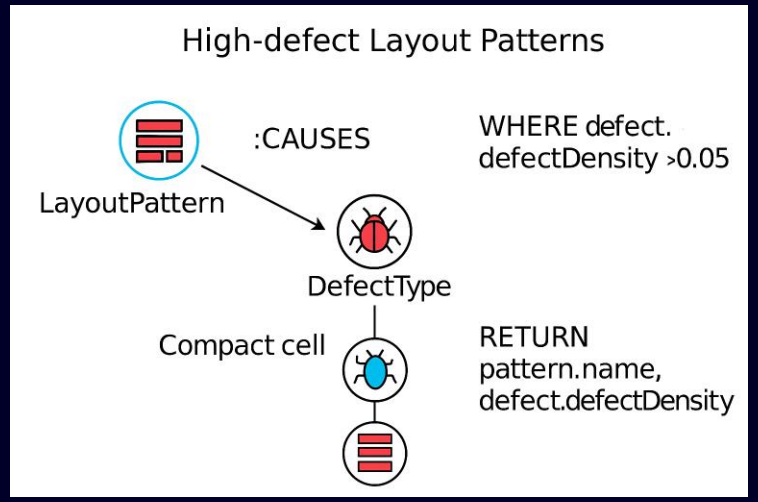
Semantic queries vs database query

Semantic Query	Database Query
Operates on **meaning and relationships** defined	Operates on **tables, rows, and columns** with no inherent understanding of meaning beyond the schema structure.
Traverses a **knowledge graph**, following edges (e.g., *path → net → layer →	Retrieves data by executing **fixed joins** across tables; cannot infer relationships unless they are explicitly modeled in the schema.
Supports **reasoning** — pattern classification, rule inference, constraint propagation — enabling queries like *“find equivalent patterns under density rules.”*	Supports **filtering and joining** concrete stored values; no built-in reasoning, inference, or semantic propagation.
Connects **cross-domain information** (design ↔ fab ↔ test	Stays within the **boundaries of the database schema**; cross-domain links require manual join definitions and are not automatically recognized.

Example Knowledge Graph queries (Independent of tools)

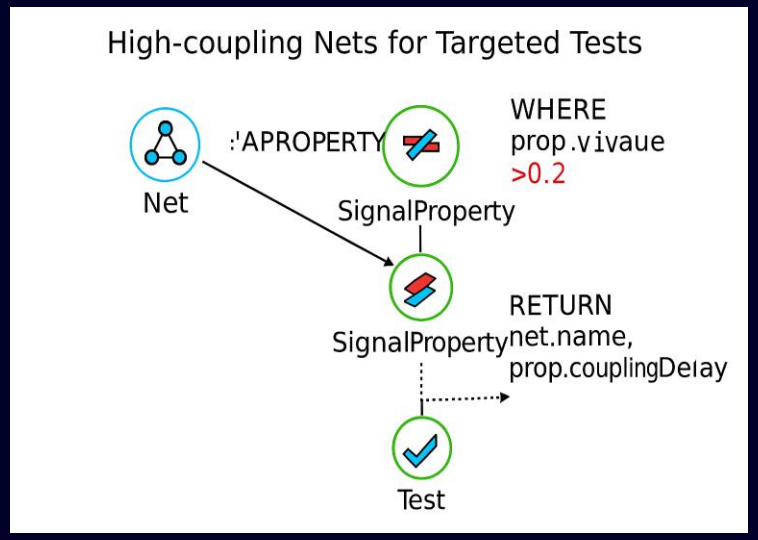
High-defect layout patterns in descending order

```
MATCH (pattern:LayoutPattern)-[:CAUSES]->(defect:DefectType)
WHERE defect.defectDensity > 0.05
RETURN pattern.name, defect.defectDensity
ORDER BY defect.defectDensity DESC;
```



High coupling nets for targeted tests

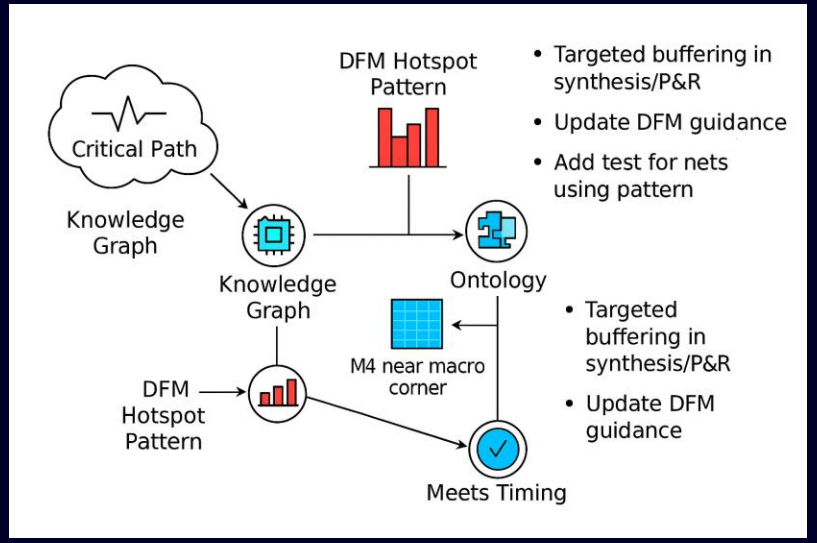
```
MATCH (net:Net)-[:HAS_PROPERTY]->(prop:SignalProperty)
WHERE prop.couplingDelay > 0.2
RETURN net.name, prop.couplingDelay
```



More examples

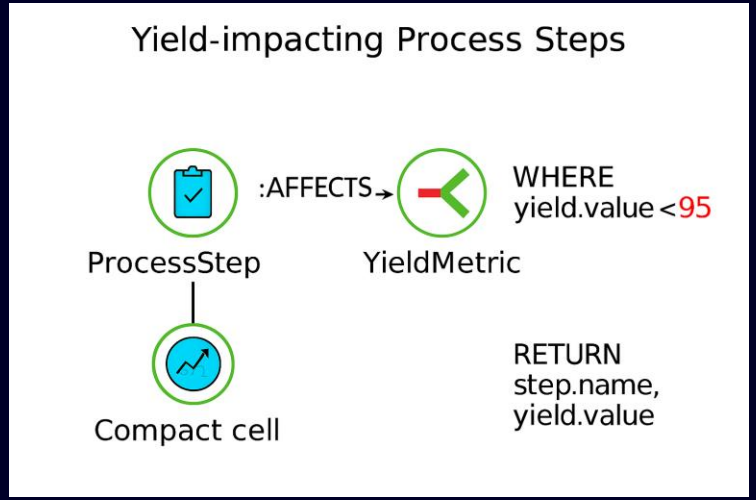
Critical nets impacted by DFM hotspots:

```
MATCH (net:Net)-[:BELONGS_TO]->(path:TimingPath),
      (net)-[:LOCATED_IN]->(layer:MetalLayer),
      (layer)-[:HAS_DFM_RULE]->(rule:DFMRule)
WHERE rule.violationCount > 100
RETURN net.name, path.slack, rule.name;
```



Yield-impacting process steps:

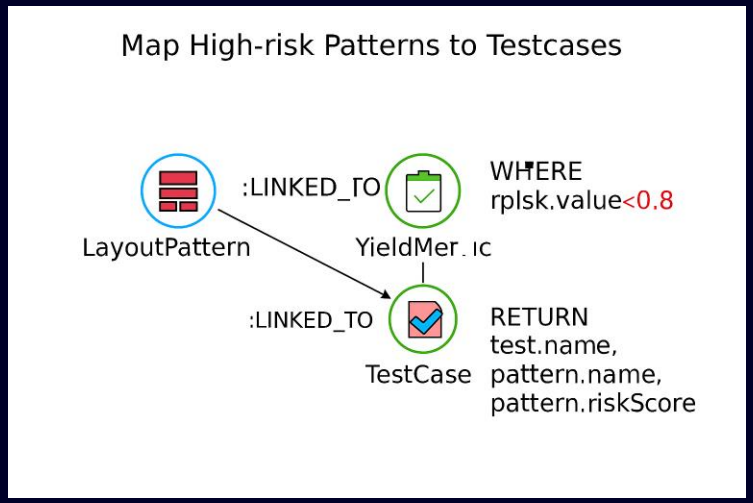
```
MATCH (step:ProcessStep)-[:AFFECTS]->(yield:YieldMetric)
WHERE yield.value < 95
RETURN step.name, yield.value;

ORDER BY prop.couplingDelay DESC;

RETURN test.name, defect.type, pattern.name;
```



More examples

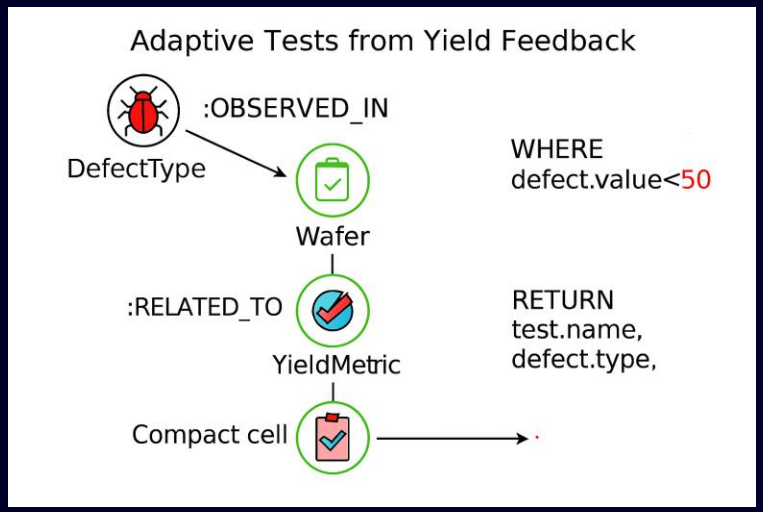
Map high-risk patterns to testcases:

```
MATCH (pattern:LayoutPattern)-[:LINKED_TO]-
>(test:TestCase)
WHERE pattern.riskScore > 0.8
RETURN test.name, pattern.name, pattern.riskScore;
```



Adaptive tests from yield feedback:

```
MATCH (defect:DefectType)-[:OBSERVED_IN]->(wafer:Wafer),
      (wafer)-[:RELATED_TO]->(pattern:LayoutPattern),
      (pattern)-[:LINKED_TO]->(test:TestCase)
WHERE defect.frequency > 50
```



Implementation considerations

- Data model: Define an ontology covering libraries (Liberty), constraints (SDC), tech files (LEF/DEF), timing/power models, and DFM rule taxonomy.
- ETL: Ingest STA reports, DEF/LEF, DRC/LVS/DFM logs, metrology and wafer sort into the KG; normalize fields to ontology terms; keep provenance for traceability.
- APIs: Expose Cypher/SPARQL endpoints; wrap common queries in CAD scripts and dashboards.
- Governance: Version ontologies, track foundry rule updates, and validate mappings in CI for sign-off flows.