WENGOLD Architecture Specification

The open-source validity kernel for silicon verification — v0.5.0

WENGOLD answers one question after every change: which of my previous results still hold? It returns **valid**, **invalidated** or **unknown**, and it never reports a pass it cannot back. The design rule is conservative by construction: empty dependency sets, unresolved nodes, dependencies absent from the graph and approximate graphs all yield unknown, so the failure mode is extra work rather than a false pass.

A result is a claim with limits, not a green tick

Every result can carry a **contract**: the **claim** it makes, its **scope** (files, configuration, reset), the **method** (kind, engine, settings), its **assumptions**, the **evidence** (fingerprints, commands, proof status, artifacts) and its **limits**. A stored pass is therefore not read more broadly than the check supports.

Four questions are answered as independent fields — **execution** (did it run), **currency** (is it still current), **establishes** (what does it prove) and **acceptance** (was it accepted). A waiver records an engineering decision and never becomes a technical pass. Reuse itself is an auditable decision: `decide()` evaluates explicit obligations and exports a decision record that `verify_decision()` re-derives independently.

Dependencies are typed, and composition is contracted

Not every dependency invalidates a result the same way, so each edge carries a kind — **compilation**, **elaboration**, **behavioral**, **assumption**, **configuration** or **requirement** — and each kind has its own reassessment rule. A changed comment does not invalidate a proof; a changed assumption does, even though the file is untouched.

A block declares what it **assumes** of its environment and what it **guarantees** to its consumer. Composition checks the two against each other, so a subsystem assembled from verified blocks is not assumed correct merely because its parts are.

Rerun the minimum, not the closure

Classification marks what a change reaches; `minimum_new_evidence()` then computes the smallest set of results that must be re-established for the graph to be sound again, taking coverage and result dependencies into account rather than re-running the transitive closure. A result already invalidated by a change is not re-run twice for the same reason, and a result whose evidence is still current is not re-run at all.

The graph is built from a real parse, not from text

`wengold index` has two backends and picks the best available. **slang** is a real SystemVerilog front-end: a preprocessor resolves `` `include ``, `` `define `` and `` `ifdef `` against the project's defines, and the source is then parsed, so modules, hierarchy instantiations, interfaces, packages, imports and parameters come from a parse tree. **regex** is the bounded elaborator, used when slang is not installed; it is a fallback, not a parser.

Anything a file does not resolve — a parse error, or an instantiation with no definition in the file set — marks the module unresolved, and the kernel then refuses to call anything downstream valid.

Measured on real designs

Measure	Before	After
Instantiations found in OpenTitan prim/rtl	195, of which 183 invented from prose	183 real
Preprocessor parse errors, same set	1504	6 (163 of 165 files clean)
OpenTitan hw/ip, 2,070 files	every `ifdef unresolved	2,555 nodes, 12,276 edges, 76 unresolved (3.6%)
CVA5, 103 files	not supported	121 design units, 2 unresolved (98.3%)

Rerun reduction, computed over the real dependency graphs above:

Change	Results re-run	Reduction
cva5 (RISC-V core top)	11 of 124	91.1%
aes_cipher_control (OpenTitan AES block)	21 of 232	90.9%

The *results* in that measurement are synthetic — no simulator ran — so it measures dependency-driven reduction, not wall-clock saving. A suite weighted toward expensive integration tests would show less.

The kernel is checked against itself

Five invariants — no upgrade under added uncertainty, known changes invalidate consumers, no confidence from removed information, determinism, and withdrawal propagation — are checked over generated graphs and against an independent reference implementation that enumerates upstream paths rather than computing a closure. `invariants.fuzz(range(400))` reports zero violations. Writing these found a real defect: a result depending on a node absent from the graph had been returning valid.

Evidence is append-only and tamper-evident

The ledger is a hash chain with locking, sequence and chain validation, malformed-record handling and a trusted checkpoint that detects truncation. Evidence is bound to its inputs, tool, arguments, seed and environment, so a stored result can be re-run rather than trusted.

Formal: prove, or say unknown

One property IR is written once and consumed by every backend — compiled to SystemVerilog assertions for SymbiYosys, and to the in-process engines:

```
Property (IR) --+--> SVA module (sby / commercial formal)
                +--> explicit engine (reachability fixpoint)
                +--> z3 engine (BMC + k-induction)
```

A bounded RTL front-end parses a documented subset into a transition function. A run that neither proves nor finds a counterexample is bounded, not a pass, and anything outside the subset is unknown. `formal/loop.py` wraps this in a counterexample-guided loop — check, diagnose, propose a repair, apply, re-check — and each property leaves a contract whose method is proof only when it really was proven.

Tool outcomes are honest by construction

Missing, timeout, crash, failed check and inconclusive proof are distinct outcomes. A formal verdict comes from SymbiYosys's own status line, not its exit code: sby can exit zero having proved nothing. The registry covers the same three groups a commercial flow is organised around — simulation and testbench, formal and FPGA, and the surrounding flow — and the open adapters were verified against real tools: verilator 5.053 lint, yosys 0.70 synthesis, SymbiYosys v0.70 (a real unbounded proof, DONE PASS) and cocotb 2.1 with Icarus Verilog 14. Running them found five defects in the adapters themselves, all fixed.

Agents and MCP

Operational agents (testgen, debugger, repair) run behind an `Orchestrator.run_loop()` that drives classify, run, evidence, diagnose and repair. The orchestrator has been run against real verilator, sby and cocotb, with the loop settling on real outcomes. An MCP server over stdio exposes the same surface to external agents.

Limits, stated plainly

- The formal front-end accepts **0 of OpenTitan's 165 prim/rtl modules**. It is a documented subset; the graph and kernel paths are what are usable on real code.
- Register maps need SystemRDL or IP-XACT. OpenTitan's primary format is HJSON (858 files) and is not yet read.
- slang is optional. Without it the regex fallback is much weaker.
- Elaboration is per file today, so CDC reports a signal driven by a submodule as having no determinable domain. Whole-project elaboration is the next step.
- Analog and mixed-signal verification and physical signoff are out of scope. Missing tools and unpinned PDKs are reported as missing, never as passes.

Tests: 160 passing with slang, 153 passing and 7 skipped without; CI runs all three configurations. **Licence:** MIT OR Apache-2.0.