Symbolic Execution of Digital Hardware Designs

A Systematic Map of Path-Conditioned Design Execution

Bili Dong[1]

Draft 2026-08-12 · landing page · survey record

Abstract

"Symbolic execution" is used broadly in hardware verification, sometimes for path exploration and sometimes for any solver-backed propagation of symbolic values. This survey adopts a narrower operational boundary: an included method executes a digital design or claimed-faithful executable representation, carries symbolic hardware values, constructs predicates for alternatives distinguished by that representation, uses feasibility to choose or reconstruct execution, and makes that mechanism load-bearing in its result. 17 broad searches through 10 August, 4 strict-boundary searches through 11 August, and critical citation chases through 12 August 2026 yield 31 full-text-qualified publication records, including preprints, after agent adjudication with no item-level human screening; 1 strict query failed and the map makes no closure or population claim. Under an explicit reproducible classification rule, 14 are classical, 15 concolic, and 2 selective-hybrid. RTL is the design target for 20; SystemC/TLM, mixed-level, netlist, another HDL, and 1 hardware-specific HLS study form the remainder. Operational artifacts and their semantic bridges are analyzed separately. Symbolic simulation, STE, BMC, and trace-only search are treated as adjacent, not counted. The corpus shows that hardware path execution is defined as much by clocking and process composition, plus scheduling where it is semantically real, reset, translation, and environment as by the solver. Target guidance, backward execution, fragment composition, caching, symbolic time abstractions, and fuzzing handoffs reduce selected work but may shift cost to paths, summaries, formulas, concrete corpora, handoffs, or semantic validation. Across the deep reads, heterogeneous bounds and harnesses prevent a quantitative ranking. The bounded map identifies a coherent, specialized slice of hardware verification. We conclude with a common reporting contract and an agenda for semantic validation, benchmarks, completion accounting, and cross-level composition.

1. Introduction

In the canonical software account, symbolic execution replaces inputs with symbols, advances a program state, forks or otherwise distinguishes branch outcomes, accumulates a path condition, and asks a solver whether that path is feasible (Baldoni et al., 2018). Moving this idea to digital hardware changes the object being executed. Branches in concurrent HDL processes contribute coupled constraints; state updates occur at clock or delta-cycle boundaries; a generated C++ model may stand between RTL and the executor; reset and an environmental testbench determine which symbolic states are reachable. A hardware "path" is therefore not merely a list of source-level branch outcomes.

Terminology obscures this point. Hardware papers also use **symbolic simulation** for evaluating a circuit over symbolic values, **symbolic trajectory evaluation** (STE) for abstract trajectories, **bounded model checking** (BMC) for solving unrolled transition formulas, and **concolic testing** for generating a new input from the symbolic predicate of a concrete trace. All can use the same SMT solver and return a counterexample. That surface overlap does not make them one method.

[1]The byline names the accountable human author, who directed the scope and takes responsibility for the manuscript. OpenAI Codex systems (through GPT-5.6 Sol and GPT-5.6 Luna) provided substantial assistance with search, blinded agent screening, evidence organization, synthesis, drafting, adversarial review, and repository tooling. AI output is not treated as evidence; literature claims rest on the cited sources. The linked survey record preserves the protocol, catalog, decisions, and evidence trail.

This survey follows path-conditioned execution. Its earliest mechanism-verified included work, rather than its earliest use of a label, is a 2011 SystemC semantics that carries symbolic stores and path conditions, checks feasibility, and composes scheduler-aware traces between waits (Harrath et al., 2011). The bounded corpus then develops through direct and translated RTL execution, concolic test generation, security-guided backward and forward search, HLS-source execution with hardware datatypes, and cross-level co-execution. This history is narrower than symbolic hardware reasoning as a whole, but it is methodologically coherent.

The survey asks five questions:

- **RQ1:** Definition and regimes — Which works actually perform classical, concolic/dynamic, or selective/hybrid symbolic execution of a digital hardware design, and how do those regimes differ operationally?
- **RQ2:** Design target and operational representation — Which hardware representation is the paper's claim about—RTL, another HDL, HLS, SystemC/TLM, a netlist, or a coupled cross-level model—which program or IR does the executor actually step, and how are concurrency and clocked state bridged between them?
- **RQ3:** Paths and scaling — What constitutes a path through concurrent, clocked hardware; how are path-conditioned alternatives enumerated, reconstructed, composed, merged, or selected; and which mechanisms control their growth?
- **RQ4:** Verification contract — Which goals are served, what witnesses or proofs are produced, and what bounds, approximations, environmental models, or harness assumptions qualify the result?
- **RQ5:** Evidence — How are systems evaluated with respect to design scale, temporal depth, coverage, solver work, defects, reproducibility, and comparison baselines?

The contributions are fourfold. First, a five-part operational test separates path-conditioned execution from adjacent symbolic methods. Second, a bounded systematic map describes the verified corpus by execution regime and design target. Third, a critical synthesis connects path meaning, hardware semantics, scaling mechanisms, and result contracts. Fourth, a compact reporting tuple makes positive witnesses, bounded completion, and inconclusive search outcomes comparable without requiring a common implementation architecture.

The principal finding is intentionally modest: the bounded map identifies a coherent, specialized slice of hardware verification rather than a synonym for formal hardware verification. Its 31-record strict corpus justifies a focused survey under this protocol; it is not a population or field-maturity estimate and is not a reason to enlarge the denominator with neighboring methods.

2. The operational boundary

2.1. Five necessary conditions

We include a work only when all five conditions hold:

1. it steps, replays, or composes executions of a digital hardware design or a claimed-faithful executable representation;
2. a hardware input, state element, or environmental value is symbolic;
3. the method constructs path predicates tied to alternatives distinguished by the executed design representation —control-indexed and, for sequential designs, time-indexed—not only by an external procedural testbench;
4. feasibility over those predicates controls enumeration, selection, reconstruction, composition, merging, or generation of another execution; and
5. this mechanism is load-bearing and yields a test, witness, coverage result, or qualified verification result.

For a translated, generated, lifted, HLS, or other derived artifact, a documented semantic relation to the design must also be adequate to the claimed result. Replaying one generated witness checks that path through the bridge; it does not establish equivalence of every source and derived execution.

A convenient symbolic state is $(\ell, \sigma, \pi, \tau, \eta)$: execution location or frontier ℓ, symbolic store σ, path predicate π, hardware time and scheduler state τ, and environment η. The tuple is descriptive rather than prescriptive. A direct SystemC executor makes scheduler choices and waits part of ℓ and τ (Harrath et al., 2011); a Verilator-to-C flow delegates them to the generated transition step and harness (Mukherjee et al., 2015; Zhang et al., 2016); a SystemC-aware KLEE extension implements its own scheduler (Lin et al., 2016). What cannot disappear is an execution identity whose feasibility affects what is executed next.

2.2. Three included regimes

Classical symbolic execution advances one or more symbolic design paths and uses their predicates directly. It includes forward, backward, and execution composed from fragments; those are search or representation choices, not separate top-level regimes.

Concolic or dynamic symbolic execution couples concrete and symbolic views of a path. A concrete RTL trace supplies branch outcomes; the method maintains or reconstructs their symbolic predicate, negates or changes a selected condition, solves it, and re-executes the resulting input (Ahmed, Farahmandi, & Mishra, 2018). Concolic execution therefore differs from classical multi-path exploration but remains symbolic execution: without the symbolic path predicate and feasibility query, the next execution would not be generated.

Selective-hybrid symbolic execution makes the symbolic executor one phase or region of a broader search. Fuzzing may provide a state snapshot from which a short symbolic suffix is constructed (Jayasena et al., 2025). The symbolic phase must still pass all five conditions. Concrete simulation plus a genetic algorithm, trace ranking, or coverage feedback alone does not.

The catalog uses one reproducible primary regime rather than conflating every concrete/symbolic handoff with hybrid search. **Selective-hybrid** is reserved for a non-symbolic search engine that maintains its own evolving frontier or corpus, can make exploration progress between symbolic invocations, and exchanges candidates with the symbolic executor. Concrete simulation or replay that only supplies and checks traces in a solve-and-replay loop remains **concolic**; remaining direct symbolic-state systems are **classical**. An LLM repair loop that consumes concolic tests is not itself an independent search engine.

2.3. What remains outside

Symbolic simulation propagates symbolic values through a circuit and may merge alternatives into guarded expressions. Historically, Carter et al. described machine symbolic simulation as similar to program symbolic execution (Carter et al., 1979); later RTL simulators used Boolean guards, ITE values, and guarded events (Kölbl et al., 2001), and dynamic functional-space partitioning split symbolic values when BDDs grew (Feng et al., 2004). These methods are relevant intellectual context. Under the present protocol they are outside unless distinguishable path predicates guide further execution.

STE, BMC, property checking, equivalence checking, theorem proving, and static information-flow analysis are likewise outside when they reason over abstract trajectories or transition formulas without a load-bearing path executor. The boundary is architectural, not evaluative: a BMC engine may prove a stronger bounded property while remaining a different technique. Surveys of general formal hardware verification and directed testing cover those broader neighborhoods (Camurati & Prinetto, 1988; Jayasena & Mishra, 2024).

Two close negative examples show why full-text adjudication matters. GreyConE concolically executes compiled SystemC, but the paper does not establish hardware-specific execution semantics or a relation to generated hardware; it is excluded at the required semantic bridge (Debnath et al., 2022). Forbench steps a symbolic RTL transition model and forks a procedural testbench while merging design alternatives into symbolic values. Its feasible alternatives are testbench-control paths, not path identities distinguished by the executed design representation, so it fails the sharpened third condition (Yang et al., 2026). Their labels are less important than the missing obligation.

2.4. What "hardware" means

Hardware names the **represented digital design**, not the implementation language and not necessarily a fabricated device. RTL is the center. Other HDLs, SystemC/TLM, HLS source or IR, generated C/C++, gate netlists, and coupled models qualify when hardware-specific semantics are load-bearing or a validated relation connects the executed artifact to hardware. Firmware symbolically executed against a concrete processor remains software symbolic execution. Generic C++ intended for synthesis remains software analysis until hardware datatypes, streams, timing, or a source-to-generated-design relation enters the method.

3. Method

We conducted a bounded systematic map followed by critical narrative synthesis. The public record contains the protocol, standing queries, complete catalog, append-only search log, source notes, evidence ledger, and

validator. The design follows systematic-mapping and backward/forward-snowballing guidance (Petersen et al., 2015; Wohlin, 2014). It does not claim that every relevant publication has been found.

3.1. Discovery and update

No domain-specific historical cutoff was imposed. APIs were queried from 1900, before the relevant digital-hardware literature. 17 broad active queries were reconciled through 10 August 2026; 4 added strict-boundary queries were reconciled on 11 August; a fifth strict Semantic Scholar query failed that day and remains unreconciled. Critical-work citation chases were run on 12 August, with defective directions disclosed below. OpenAlex, Crossref, arXiv, and Semantic Scholar searches combined **symbolic execution**, **dynamic symbolic execution**, **concolic**, **path condition**, and **path constraint** with RTL, Verilog, SystemC, HLS, processor, hardware security, and hardware verification terms. Backward and forward citation searches supplemented database discovery. When an index omitted a plausible bibliography, the primary paper's reference list was screened.

The initial broad campaign reconciled 1,262 deduplicated database-search records, 273 chase-only records, 41 Carter-lineage records, 48 Forbench bibliography records, and 2 directly inspected primary additions, yielding a baseline of 1,626. After the scope was narrowed, the revision attempted 5 boundary-focused queries: 4 returned result sets, and 1 failed with HTTP 429. The four successful exports and two primary-version additions contributed 83 new catalog records after source aliases were collapsed.

The stricter classification also promoted a new set of works to critical. Both citation directions were attempted for each. Index and primary-paper chases added 161 records after overlap reconciliation. An adversarial audit found that their initial exclusion partition had been produced by title rules; all 161 were subsequently re-screened as individual metadata judgments. A delta review rejected 138 still-template-shaped rationales; three reviewers then re-adjudicated those records in bounded item batches against retained metadata, with close cases checked against available abstracts and primary records. The catalog now records one work-specific rationale per chase record. The public audit record preserves the exact 138-record repair subset and the original and final disposition fields for all 161 records. Printed bibliographies for the two zero-index seeds added 44 further records. No chase-only work changed the strict include set. Forward discovery for AutoVeriFix+ and the HLS thesis remains incomplete: the index returned no usable forward records, and no zero-result event is treated as evidence of coverage. Two older provenance claims were also narrowed. Carter's backward event resolves only part of its claimed record set, and the legacy Sylvia event's screened total lacks a complete key partition; neither is now called complete. The failed Semantic Scholar query remains registered and unreconciled rather than being treated as an empty result. Finally, 2 methodology works cited to document mapping and snowballing were added as excluded secondary context when their local survey notes were made self-contained. Binding the Sylvia external decision home at key level also added 1 previously absent boundary comparator from that delegated bibliography. The baseline and all 5 addition batches therefore sum to the 1,917-record catalog.

3.2. Selection and coding

The broad baseline was screened in two blinded title/abstract passes with item-specific reasons, followed by adjudication of every decision or code disagreement. The strict revision then re-adjudicated every previous include, every previous parked record, and every strict-query candidate; aliases were reconciled to existing records. A positive decision required primary full text supporting every part of the five-part operational test. A clear primary abstract could support exclusion; an otherwise plausible record without decisive full text was parked. Duplicate or superseded versions remained in the catalog under E8 so the denominator would not silently shrink. A later provenance audit appended the keys omitted from the original strict-reclassification event rather than rewriting history. For historical search and chase rows that compacted away overlaps, corrective audit rows now record every raw screened position as a canonical key, including repeated keys when two source records collapse to one work.

Each include has one value on four substantive facets. Together they identify the design claim, primary execution regime, sought result, and evaluation form:

Facet	Values
Design target	RTL; other HDL; HLS; SystemC/TLM; gate netlist; mixed level; generic
Primary regime	classical; concolic; selective-hybrid
Goal	functional; test coverage; security; equivalence; general
Evidence	experiment; case study; formal only; none

Table 1: Coding facets for the strict include-level corpus.

"Deep-read" denotes the 17 load-bearing works with a local, survey-specific source note. The remaining includes were still checked against primary full text for eligibility but retained at mapping depth; detailed algorithm and performance claims do not rest on them.

3.3. Bounded map result

The catalog contains 1,917 records. Of these, 31 are include-level publication records: 17 were deep-read and 14 were retained at mapping depth. A further 59 are parked, while 1,827 are excluded with a declared reason. The 31 include preprints and were agent-adjudicated without item-level human screening. They are publication records, not independent tool implementations, a population estimate, or a mapping-closure claim.

Under the classification rule, the execution map contains 14 classical, 15 concolic, and 2 selective-hybrid works. By design target, 20 claim about RTL, 4 about SystemC/TLM, and 4 about coupled mixed-level models. The remaining targets comprise 1 other-HDL work, 1 HLS-source work, and 1 gate-netlist work. Operational artifacts such as generated C/C++ and lifted IR are analyzed separately because they do not disappear behind the target label.

The cross-tabulation makes the concentration inspectable:

Design target	Classical	Concolic	Selective	Total
RTL	7	11	2	20
SystemC/TLM	2	2	0	4
Mixed level	3	1	0	4
Other HDL	0	1	0	1
HLS	1	0	0	1
Gate netlist	1	0	0	1
Total	**14**	**15**	**2**	**31**

Table 2: Primary execution regime by design target. Values are machine-checked against the catalog.

The catalog itself is the canonical work-by-facet map. Section 12 renders all 31 rows with full title, target, regime, goal, evidence class, and scrutiny depth.

3.4. Validity of the agent-assisted procedure

The two initial screens used related AI systems, so their errors may be correlated; the accountable human author directed the scope but did not perform item-level screening. Stronger adjudication, retained canonical decisions, full-text notes for the critical set, and machine-derived counts improve auditability but are not independent replication. The initial classifier-produced chase partition was discarded and every affected item was re-adjudicated from its own record; this repair strengthens traceability but does not turn related agent judgments into independent human review.

4. Corpus map and development

The strict corpus begins with a scheduler-aware SystemC construction, not with the older symbolic-simulation papers excluded by the operational test. It then grows through several partly independent applications. The chronology below is a mechanism map, not a claim that every paper directly descends from its predecessor.

4.1. Classical execution

The earliest verified include builds waiting-state automata from feasible SystemC symbolic paths (Harrath et al., 2011). By 2015–2016, software analyzers were being applied through synthesis-semantic Verilog-to-C translation and SystemC-aware or Verilator-generated execution (Lin et al., 2016; Mukherjee et al., 2015; Zhang et al., 2016).

These architectures establish a recurring choice: implement HDL scheduling in the executor, or trust a translation and execute its paths.

Security work broadened both direction and output. Among the deep reads, Coppelia uses backward execution to generate replayed processor-level exploit witnesses (Zhang et al., 2018). Later deep-read systems explore coupled hardware/software paths, gate-level information flows, statically guided RTL flows, processor/ISS mismatches, and independently constructed RTL fragments (Bruns et al., 2023; Fowze et al., 2022; Mukherjee et al., 2020; Ryan et al., 2023; Ryan & Sturton, 2023). The latest boundary expands to hardware-specific HLS source and cross-level SystemC peripherals (Hu, 2024; Rudkowski et al., 2026).

4.2. Concolic execution

The concolic line centers concrete traces as the path-selection mechanism. The catalog places mapping-depth publications on qualifying-event search, factored RTL testing, SystemC testing, multi-target activation, and selective SystemC testing in this chronology (Ahmed & Mishra, 2017; Lin et al., 2018; Lin et al., 2020; Lyu et al., 2019; Pinto, 2017). Technical synthesis here rests on the deep-read directed RTL work: it selects a target-related branch, alters the trace predicate, solves, and replays (Ahmed, Farahmandi, & Mishra, 2018).

Subsequent mapping records extend the chronology to asynchronous-reset security, equivalence coverage, and incremental RTL testing (Meng et al., 2021; Roy & Chaki, 2022; Witharana et al., 2024). The chronology also contains trace-restricted, Trojan-activation, assertion-targeting, and interleaved HDL studies (Ahmed, Farahmandi, Iskander, et al., 2018; Bagri, 2015; Qin & Mishra, 2014; Witharana et al., 2021). The deep-read scalable RTL study documents hardware-aware guidance and reuse (Lyu & Mishra, 2021). AutoVeriFix+ embeds the mechanism inside an LLM-driven RTL repair workflow: it records cycle-indexed paths, negates an uncovered branch predicate, solves a test, and re-simulates it (Tan et al., 2026).

4.3. Selective-hybrid execution

Selective hybrids reserve symbolic execution as an intervention in an independently progressing search. A mapping-depth fuzzing/concolic record marks that chronology (Debnath et al., 2021); its detailed mechanism is not used in the synthesis. FuSS provides the deep-read case: a fuzzing plateau selects a nearby RTL-CFG target, a Verilated state snapshot supplies the prefix, and a symbolic suffix returns a new program to the corpus (Jayasena et al., 2025).

4.4. What the bounded map establishes

The bounded corpus spans direct semantics, translation-based reuse, testing, security, equivalence, HLS, and cross-level checking. Those recurring themes make a focused survey useful. The map is relevance-capped, has parked records and an unreconciled query, and makes no population or maturity inference. Enlarging it with every symbolic simulator or BMC paper would change the common operational question rather than improve this map's closure.

5. Executed artifacts and semantic bridges

The catalog's design-target facet names what the paper claims about; the operational artifact names what the symbolic engine interprets. Both matter. Language labels alone are insufficient: a C++ program can be the operational representation of RTL, while another C++ program is merely source intended for synthesis.

5.1. Direct HDL execution

Direct executors parse or elaborate an HDL and define steps for expressions, assignments, branches, state updates, and scheduling. Harrath et al. make the SystemC scheduler, runnable processes, signal updates, notifications, waits, and delta cycles part of symbolic transition construction (Harrath et al., 2011). SESC similarly extends KLEE with a SystemC scheduler, signals, FIFOs, arbitrary-width integers, and clocks (Lin et al., 2016). Direct RTL systems make source locations and path predicates easy to relate, but inherit a substantial conformance obligation: nonblocking assignment, event ordering, memories, four-state values, multiple clocks, and unsupported testbench constructs can all change the concrete trace relation.

5.2. Generated executable representations

Translation-based systems reuse software symbolic executors. V2C lowers synthesizable Verilog to a word-level C transition program and then selects a path-symbolic-execution analyzer distinct from its BMC and abstract

interpretation configurations (Mukherjee et al., 2015). SE4RDV uses Verilator, an arbitrary-bit-width KLEE variant, and a harness whose `eval()` loop defines the cycle horizon (Zhang et al., 2016). Their trusted chain is

$$\text{design} \rightarrow \text{generated program} \rightarrow \text{symbolic executor}.$$

Translation removes the need to reimplement all HDL scheduling, but does not remove semantics. Widths, undefined behavior, generated control flow, memory, clock sequencing, and harness assumptions must preserve the behavior relevant to the claim. Replaying a generated test on the source design validates that one witness; it is not a proof that all translated paths correspond.

5.3. SystemC/TLM and mixed levels

SystemC combines C++ with a simulation kernel, process scheduling, events, and time. A method that merely compiles a SystemC-looking function is therefore not equivalent to a SystemC executor. Cross-level peripheral checking makes the issue explicit: one symbolic run may combine a low-level implementation with a transaction-level reference, and the solver observes their disagreement (Rudkowski et al., 2026). Unsupported asynchronous waits or context-switch approximations limit the explored behaviors.

Other mixed-level systems couple different semantic objects. COVERIF explores hardware/software interaction paths (Mukherjee et al., 2020). The RISC-V case study co-executes Verilated processor RTL and an instruction-set simulator, comparing them at retired-instruction boundaries (Bruns et al., 2023). Such oracles can expose interface or implementation discrepancies, but agreement is only relative to input restrictions, timing alignment, observation points, and the reference model.

5.4. Netlists and HLS

EISec translates a sequential gate-level netlist into a C representation for KLEE, making state initialization and netlist-to-C fidelity central to its information-flow results (Fowze et al., 2022). Its under-constrained state model can surface possible flows that are not reset-reachable; that is useful exploration but a weaker deployment claim.

The single included HLS study is deliberately qualified. Hu extends KLEE for Vitis `ap_uint`, `hls::stream`, concurrent stream behavior, and clock-linked state in a hardware TCP stack (Hu, 2024). Hardware semantics are therefore load-bearing, so the thesis passes the boundary. It does not validate the generated RTL, so its tests and bugs establish properties of the HLS source model, not automatically the synthesized circuit. Intended synthesis alone would not have been enough for inclusion.

6. Execution regimes and path meaning

6.1. Classical paths

A classical executor advances states such as $(\ell, \sigma, \pi, \tau, \eta)$. At a symbolic guard g, successor predicates are $\pi \wedge g$ and $\pi \wedge \neg g$; infeasible successors are removed. In clocked hardware, τ includes cycle or scheduler state, so the same source branch reached on two cycles can belong to different paths. In concurrent SystemC, process selection, event notification, and wait boundaries can also distinguish executions (Harrath et al., 2011; Lin et al., 2016).

Generated-code systems expose the paths selected by a compiler or simulator rather than the HDL syntax directly (Mukherjee et al., 2015; Zhang et al., 2016). This can be entirely adequate for test generation if witnesses replay, but source branch coverage and generated C++ branch coverage need not have the same denominator.

Classical does not require forward whole-path enumeration. Coppelia reasons backward from an architectural security condition and stitches cycle-level predecessors into a replayed program (Zhang et al., 2018). Sylvia explores sequential RTL blocks independently and asks whether fragment combinations form a feasible design execution (Ryan & Sturton, 2023). Both retain path identity and feasibility; direction and fragmentation are scaling choices within the regime.

6.2. Concolic paths

Concolic execution represents the active path twice: a concrete trace fixes the branch sequence and hardware state, while symbolic constraints explain which inputs could reproduce or divert it. Directed RTL testing selects a branch related to a target, negates its predicate, solves the prefix, and re-simulates the returned vector (Ahmed, Farahmandi, & Mishra, 2018). Scalable variants add hardware-aware ranking, caching, and reuse (Lyu & Mishra, 2021).

This is narrower than classical multi-path execution but not outside symbolic execution. The solver operates on a predicate tied to an executed path, and its model causes the next hardware execution. The distinction matters for claims: only traces selected by the concrete schedule and diversion policy are considered, so exhausting the currently selectable alternatives is not a general reachability proof.

AutoVeriFix+ demonstrates that the same mechanism can be nested in a larger application. Concrete simulation records cycle-indexed branch and state traces; the concolic phase changes an uncovered branch; differential checking and an LLM use the new trace for repair (Tan et al., 2026). The generated Python oracle and repair loop qualify the correctness claim, but they do not change the concolic classification.

6.3. Selective-hybrid paths

Selective hybrids distribute exploration state across independently progressing engines. FuSS holds a concrete program corpus and coverage map, snapshots the Verilated design state at a selected frontier, and symbolically solves only a short suffix (Jayasena et al., 2025). The path is a composite of concrete prefix and symbolic diversion.

A hybrid result is the union of its phases, not the strongest property of the symbolic phase. Target selection, slicing, state reconstruction, and concretized choices determine which behaviors can be missed. A complete description therefore names the handoff trigger, the state transferred, and whether an earlier concrete choice can later become symbolic.

For a reproducible partition, the additional engine must maintain its own evolving frontier or corpus, make progress between symbolic invocations, and exchange candidates with the symbolic executor. A simulator that merely supplies and replays one trace in a solve-and-replay loop does not qualify; those systems, including Qin and Mishra, remain concolic. Classical covers the remaining direct symbolic-state systems.

6.4. Concurrency and time

Hardware paths have at least three axes: control choices within a process, composition or interaction among processes, and repeated state transitions over time. Ordinary synchronous RTL processes denote coupled same-cycle behavior, not arbitrary scheduler choices. Scheduler paths can be semantically real in a SystemC kernel, asynchronous events, multi-clock interactions, or a racy testbench; some tools instead serialize concurrency as an implementation choice or approximation. These models can agree for a supported subset and diverge elsewhere. Path count has meaning only after the concurrency, clock, reset, and environment models are stated.

7. Scaling path-conditioned hardware execution

If each of n effective binary choices is independent, explicit path exploration can expose 2^n executions. Hardware supplies new choices every cycle and can combine independently controlled processes, symbolic addresses, environment transactions, and state-dependent loops. Practical systems therefore decide which paths to construct now, which constraints to reuse, and which prefix or suffix to leave concrete.

7.1. Guidance and one-path reconstruction

Directed concolic testing ranks candidate branch diversions by their relation to a target rather than enumerating paths uniformly (Ahmed, Farahmandi, & Mishra, 2018). Scalable RTL concolic testing strengthens this with contribution analysis and reuse (Lyu & Mishra, 2021). SEIF uses a static information-flow graph as an overapproximate route, segments it at clock boundaries, and performs bounded symbolic search for an executable RTL witness (Ryan et al., 2023). These mechanisms improve time to selected evidence; unless every target and diversion is scheduled to completion, they do not establish absence.

Reconstructing only an observed path also limits formula size. Qin and Mishra specialize dynamic array indices to a trace and reuse unsatisfiable cores (Qin & Mishra, 2014). The saving comes with an obligation: constraint deletion and trace specialization must preserve the branch diversion being claimed.

7.2. Backward search, fragments, and summaries

Backward execution avoids enumerating every reset-to-target prefix by starting from a security condition and constructing predecessor cycles (Zhang et al., 2018). Fragment-based execution explores blocks or modules independently before solver-checked composition (Ryan & Sturton, 2023). Both postpone part of the global product. Their summaries must retain enough state, time, and interface conditions that a stitched path corresponds to a realizable execution.

As a direct illustrative worst-case derivation, consider N blocks with b independent local binary choices. Building local trees can cost on the order of $N \cdot 2^b$ rather than constructing 2^{Nb} whole paths immediately. The compatible fragment tuple space can still reach 2^{Nb}. Construction reduction is valuable, but it should not be reported as elimination of the composition product.

7.3. Time abstraction and selective suffixes

HLS TCP execution replaces many concrete idle cycles with a symbolic packet gap, reducing repeated clock paths while adding an abstraction obligation (Hu, 2024). FuSS uses a concrete fuzzer for long prefixes and asks a solver for only a nearby CFG suffix (Jayasena et al., 2025). Both succeed by preserving the distinctions important to their observer while compressing or concretizing others.

Hybrid handoffs add their own costs: detecting a plateau, mapping coverage to a target, reconstructing a faithful state, solving the suffix, and replaying the resulting test. A local solver speedup can be outweighed by these phases, while an unreachable snapshot can yield a nonreplayable witness.

7.4. Four accounting ledgers

Ledger	Growth	Representative controls
Executor	active paths, fragments, cycles	guidance, backward search, composition
Formula	predicate size, aliasing, theory	slicing, caching, incremental solving
Concrete frontier	traces, seeds, coverage state	ranking, fuzzing, handoff policies
Semantic bridge	translation, replay, harness work	supported subsets, differential validation

Table 3: Four ledgers for checking the end-to-end effect of a local scaling mechanism.

This is a cost-accounting lens, not a conservation principle or impossibility theorem. Tools can genuinely reduce work by quotienting behavior irrelevant to an observer, reusing stable summaries, or spending solver effort only at difficult boundaries. A local gain may also shift work to another ledger, so a credible claim reports all four, end-to-end time and memory, completion status, and the distinctions deliberately omitted. No numerical quantity is asserted to be conserved.

8. Verification goals and result contracts

The same path engine can generate tests, close coverage, expose an exploit, compare two models, or check an assertion. The application label does not determine the strength of the result. Output type, environment, and completion do.

8.1. Replayable positive witnesses

A concrete input sequence replayed on the design is a portable conclusive positive outcome. SE4RDV solves generated-code paths and measures the tests again in RTL simulation (Zhang et al., 2016). Directed concolic testing observes the target during concrete RTL re-execution (Ahmed, Farahmandi, & Mishra, 2018). FuSS returns symbolically extended programs to the fuzzer and measures them on the Verilated design (Jayasena et al., 2025). Coppelia goes further by turning a processor-design condition into an executable program and replaying exploits (Zhang et al., 2018).

Replay establishes an existential claim: under this reset, clock, environment, and implementation, the supplied input exhibits the behavior. It does not prove translation equivalence, validate every symbolic intermediate state, or show that another behavior is absent. A portable witness must include the initialization and transaction protocol, not only the symbolic bytes.

8.2. Coverage and repair evidence

Branch, statement, condition, assertion, information-flow, and toggle coverage have different denominators. Generated C++ branches may not correspond one-to-one with RTL branches, and excluded or unreachable targets can inflate or depress a percentage. A coverage claim therefore needs the instrumented artifact, denominator, reachability policy, temporal budget, and replay rate.

Concolic and selective-hybrid work commonly uses coverage as a target or handoff signal and as the outcome. Qin and Mishra divert concrete traces toward uncovered HDL branches (Qin & Mishra, 2014); FuSS invokes symbolic

suffix execution at a fuzzing plateau (Jayasena et al., 2025). AutoVeriFix+ uses concolic branch discovery to improve differential tests and propose RTL repair or pruning (Tan et al., 2026). None of these mechanisms turns a residual uncovered branch into a proof of unreachability or semantic redundancy.

8.3. Bounded completion and negative outcomes

Exhaustion by a sound executor can establish all modeled executions only within its initial-state, time, environment, supported-language, translation, approximation, and solver contract. An empty worklist alone is not a soundness argument. Timeouts, memory limits, solver limits, heuristic exhaustion, and targets never scheduled are incomplete outcomes even when no counterexample appears.

SEIF is unusually explicit: a static candidate can be globally contradictory, fail only under a bounded search, be rejected by a semantic-flow check, yield a replayable path, or remain unaccounted (Ryan et al., 2023). EISec's under-constrained initial netlist state broadens possible information flows but can admit states not reachable from reset (Fowze et al., 2022). Such partitions should replace the binary "found/not found" reporting common in testing papers.

8.4. Equivalence and coupled-model claims

Cross-level results need a relation between states, time, and observations. COVERIF makes a coupled hardware/software path the execution object (Mukherjee et al., 2020). The RISC-V study compares Verilated RTL with an ISS at retirement boundaries (Bruns et al., 2023). Cross-level SystemC checking uses a transaction-level peripheral as an oracle for a lower-level implementation (Rudkowski et al., 2026). A mismatch is a useful witness; agreement is only as strong as the reference model, alignment, assumptions, and completed bounds.

The same qualification applies at the HLS edge. Hu's generated tests expose failures in the hardware-specific HLS source model, but without generated-RTL validation they do not establish the behavior of every synthesis result (Hu, 2024).

8.5. Minimum reporting tuple

Every reported result should be readable as

$$(A, S, T, E, X, R, C),$$

where A is the executed artifact and translation chain; S the reset or initial state; T clock, scheduling, and temporal bound; E the environment and harness; X semantic conformance, exactness, abstraction, concretization, and supported subset; R the returned witness, coverage, or conclusion; and C whether search merely terminated or achieved justified bounded exhaustion. For a combinational design, temporal fields may be "not applicable"; they should not silently disappear. "Exhaustive," "safe," or "verified" is interpretable only with the complete tuple.

9. Evaluation practice and evidence strength

The 17 deep-read studies demonstrate functioning implementations and reported positive outcomes on their evaluated configurations. They do not establish implementation correctness, general effectiveness, or a single quantitative ranking. Designs, horizons, coverage definitions, setup effort, baselines, and completion policies differ too much for a defensible meta-analysis.

9.1. Range of evidence

Early translation-based work is feasibility evidence. V2C compares path-symbolic execution with BMC and abstract interpretation on selected hardware properties (Mukherjee et al., 2015). SESC evaluates several small SystemC designs with path/test counts, coverage, time, and memory (Lin et al., 2016). SE4RDV reports about 4,010 tests and 98.4 percent statement and 96.8 percent branch coverage on one OpenCores FPU, without a competing test-generation baseline (Zhang et al., 2016). These results establish workable architectures, not industrial generality.

Among the deep reads, directed concolic work provides multi-design comparative evidence. Ahmed et al. measure iterations, time, and memory against bounded model checking and two concolic strategies across benchmark targets, while preserving an explicit cycle bound (Ahmed, Farahmandi, & Mishra, 2018). Scalable concolic testing broadens designs and guidance comparisons (Lyu & Mishra, 2021). AutoVeriFix+ reports near-complete branch coverage on generated RTL benchmarks, but the preprint's "exhaustive" language exceeds its timeout, state-explosion, and generated-oracle caveats (Tan et al., 2026).

Several security deep reads have compelling positive outcomes. Coppelia reports 29 of 31 known vulnerabilities rediscovered and four new processor-design vulnerabilities, with generated exploits replayed (Zhang et al., 2018). SEIF measures accounted, witnessed, rejected, and unaccounted information-flow paths over four designs; one full source-signal analysis lasts 3.5 days (Ryan et al., 2023). EISec reports possible netlist-flow witnesses, but its under-constrained initialization and translation limit the word "exhaustive" (Fowze et al., 2022).

Concolic and selective-hybrid execution demonstrate complementarity rather than completeness. Qin and Mishra compare trace-reconstructed solving with random testing and a prior hybrid on bounded designs (Qin & Mishra, 2014). FuSS reports branch and toggle coverage trajectories on four RISC-V SoCs (Jayasena et al., 2025). The HLS TCP thesis generates 67 symbolic tests for 47.47 percent source coverage, nearly the same final percentage as 2,150 random tests, while also exposing specific failures and reducing paths with a clock abstraction (Hu, 2024).

Among the 17 deep reads, the cross-level SystemC study reports the widest combination observed here of evaluation dimensions: two implementations, four dual-level peripherals, functionality and interface tests, 357 selected mutants, ablations, a prior-tool comparison, and five larger modules (Rudkowski et al., 2026). Its negative evidence is important: many larger cases hit 24-hour, memory, or solver-query limits. Paths at timeout measure progress, not verification coverage. The RISC-V co-execution case similarly finds real mismatches but includes a run lasting 586,905 seconds (Bruns et al., 2023).

9.2. Recurrent comparability failures

First, **scale** is not a scalar. Source lines, gates, registers, branches, processes, and generated instructions measure different structures; all must be paired with temporal depth. Second, **completion** is blurred: feasible, infeasible, timed-out, unaccounted, and never-scheduled paths need separate columns. Third, **baselines solve different contracts**: simulation samples, concolic search pursues branch diversions, and BMC asks a bounded property query. Runtime alone does not equalize their outputs.

Fourth, **human and semantic effort** is largely absent. Testbench construction, reset modeling, target annotation, translation repair, abstraction, and reference-model debugging can dominate adoption cost. Fifth, stochastic systems in the reviewed evidence omit repeated-run dispersion and seed policy. Finally, artifact availability and semantic conformance are rarely strong enough to separate a search failure from a modeling failure.

9.3. Recommended evaluation bundle

A reusable result should publish design/version and license; structural scale; executed representation and translation commands; reset, clock, and environment harness; symbolic inputs and temporal depth; target denominator; feasible, infeasible, timed-out, unaccounted, and unscheduled outcomes; executor, solver, and end-to-end time; peak memory and query count; replay rate; stochastic seeds and dispersion; and human setup effort. The seven-element result tuple from Section 8 should accompany these metrics. This bundle enables like-for-like comparison without pretending that all tools answer the same verification question.

10. Interpretation and research agenda

10.1. Why the mapped niche is technically demanding

The bounded corpus does not establish the size or maturity of an underlying field. It does expose technical pressures that recur across the deep reads. Branch choices recur across cycles and interact with concurrency and environment; an RTL executor needs language and same-cycle composition semantics, while SystemC can add scheduler semantics and a translated executor inherits a trust and traceability gap. These are interpretations of the mapped mechanisms, not a causal publication-volume study.

The technique is especially useful when a concrete witness matters and the target is narrow: a hard RTL branch, Trojan trigger, processor exploit, information-flow path, assertion, or cross-level mismatch. Directed and scalable concolic systems make this niche explicit (Ahmed, Farahmandi, & Mishra, 2018; Lyu & Mishra, 2021). Coppelia and SEIF show the value of turning an abstract security concern into an executable path (Ryan et al., 2023; Zhang et al., 2018). FuSS shows how symbolic execution can be useful as selective assistance rather than the sole engine (Jayasena et al., 2025).

These cases explain why the bounded slice is coherent: each uses path feasibility to obtain selected evidence under a hardware semantic contract. A survey can make that niche legible without inferring field maturity or padding the denominator.

10.2. Findings summary

Finding	Evidence scope	Qualification	Consequence
Operational boundary	Full-text deep reads plus named symbolic-simulation and testbench-path comparators	Classifies native design-path execution, not tool value or solver use	Require design-distinguished predicates and feasibility-guided execution
Semantic bridge	Direct HDL/SystemC, translated RTL, netlist, HLS, and coupled-model studies	Replay validates a witness, not every source/derived correspondence	Report target and operational representation separately
Execution regimes	Classical, concolic, and two independently progressing selective hybrids	Concrete replay alone remains concolic; handoffs can omit behavior	Classify by the engine that owns the evolving frontier
Scaling	Guidance, reconstruction, fragments, caching, time abstraction, and fuzzing suffixes	Local reductions can move work into formulas, compatibility, corpora, or validation	Account for executor, formula, concrete-frontier, and bridge costs
Result strength	Recurring replayable tests and witnesses among the 17 deep reads	Absence needs sound bounded completion; heterogeneous experiments prevent ranking	Publish the full result tuple and separate witnesses from incomplete search

Table 4: Principal findings, their evidence scope, and the qualifications that travel with them.

10.3. Semantic conformance

Direct executors need executable suites for HDL widths, four-state values, nonblocking assignments, memories, clocks, reset, scheduling, and process interactions. Translation-based systems need differential replay and declared preservation boundaries. The SystemC work especially shows that C++ execution without a scheduler model is not enough (Lin et al., 2016; Rudkowski et al., 2026). Conformance failures should be reported separately from path-search failures.

10.4. Common benchmarks and outcomes

Future evaluations need versioned RTL, SystemC, HLS, and mixed-level tasks with complete harnesses, reachable and unreachable targets, expected witnesses, and several temporal horizons. Hard negative cases and timeouts should remain in the suite. Outcome schemas should distinguish proved infeasible, boundedly unexplored, abstraction-inconclusive, nonreplayable, and not scheduled. SEIF's explicit unaccounted class is a useful precedent (Ryan et al., 2023).

10.5. Compositional and incremental execution

Fragment, module, and cycle summaries should name state, time, assumptions, and observables. Research should separate the cost of constructing reusable local paths from checking global compatibility. Cross-level systems add a second composition problem: aligning implementation and reference semantics (Bruns et al., 2023; Rudkowski et al., 2026). Incremental solvers and cached path predicates are promising only when their reuse keys include the hardware history that affects feasibility.

10.6. Better concrete–symbolic handoffs

Concolic and selective-hybrid execution need principled handoff contracts: why a target is chosen, which concrete state is transferred, how reachability from reset is preserved, what slice is symbolized, and whether the output replays. FuSS provides a concrete snapshot/suffix architecture (Jayasena et al., 2025); Qin and Mishra provide

trace-derived constraints in a solve-and-replay loop (Qin & Mishra, 2014). Future work should report the cost and failure modes of reconstruction, not only solver time after the handoff.

10.7. HLS and language breadth

One HLS-source thesis cannot support a general HLS conclusion. Its hardware-specific datatype, stream, and timing work shows the right inclusion logic, while its lack of generated-RTL validation exposes the missing bridge (Hu, 2024). VHDL, Chisel, Bluespec, HLS IRs, multi-clock RTL, and broader SystemVerilog remain search and research targets. New work should either model their hardware semantics directly or validate the relation to the generated artifact.

10.8. Evidence and effort

Future evaluations should measure harness and model-building effort alongside runtime. Repeated stochastic trials, public artifacts, independent replay, semantic conformance tests, and full outcome partitions would do more for comparability than another isolated coverage percentage. AutoVeriFix+ also raises a new oracle problem: an LLM-generated reference can guide useful concolic tests while remaining an uncertain specification (Tan et al., 2026). Oracle provenance belongs in the result contract.

11. Limitations

Terminology is the main discovery limitation: relevant papers may say test generation or formal simulation, while unrelated papers may adopt the language of symbolic execution. Multi-index searches, citations, and mechanism review reduce but cannot remove this risk. 1 strict Semantic Scholar query failed; the overlapping concepts were searched elsewhere, but unique records may remain missing.

Full-text inclusion reduces false positives at the cost of false negatives when old or inaccessible texts cannot be obtained. 59 such or otherwise unresolved records are parked. The operational boundary is also a researcher judgment. Each include therefore has an item-specific rationale, and each close negative names the failed condition, but another defensible protocol could draw the boundary differently.

Historical search rows recorded canonical new decisions rather than every raw overlap. Corrective audit rows reconstruct raw positions where retained staging made that possible. Carter's backward event still resolves only part of its claimed record set, and the legacy Sylvia event's screened total has no complete recoverable key partition; neither is treated as complete. Forward discovery for AutoVeriFix+ and the HLS thesis also remains incomplete because their citation-index seeds returned no usable records.

Finally, the map codes publications rather than independent tools, retains 14 works only at mapping depth, and does not normalize designs, time bounds, coverage denominators, harness effort, or completion rules across evaluations. It therefore supports a qualitative technical synthesis and a reproducible facet map, not a population estimate, effect-size meta-analysis, or tool ranking.

12. Include-level corpus map

This table exposes the complete 31-publication map behind the aggregate matrix in Section 3. "Target" names the hardware representation about which the paper makes its claim; it need not be the executor's operational input. "Depth" separates the 17 source-anchored technical readings from the 14 mapping-only chronology and facet records.

Year	Publication	Target	Regime	Goal	Evidence	Depth
2011	Building SystemC waiting state automata	systemc-tlm	classical	functional	case-study	deep
2014	Scalable Test Generation by Interleaving Concrete and Symbolic Execution	hdl-other	concolic	test-coverage	experiment	deep
2015	Hardware Verification Using Software Analyzers	rtl	classical	functional	experiment	deep
2015	Improving Branch Coverage in RTL Circuits with Signal Domain Analysis and Restrictive Symbolic Execution	rtl	concolic	test-coverage	case-study	map
2016	Automatic generation of high-coverage tests for RTL designs using software techniques and tools	rtl	classical	test-coverage	case-study	deep
2016	Generating high coverage tests for SystemC designs using symbolic execution	systemc-tlm	classical	test-coverage	experiment	deep
2017	QUEBS: Qualifying Event Based Search in Concolic Testing for Validation of RTL Models	rtl	concolic	test-coverage	experiment	map
2017	RTL Functional Test Generation Using Factored Concolic Execution	rtl	concolic	test-coverage	experiment	map

Year	Publication	Target	Regime	Goal	Evidence	Depth
2018	A recursive strategy for symbolic execution to find exploits in hardware designs	rtl	classical	security	case-study	map
2018	Concolic testing of SystemC designs	systemc-tlm	concolic	test-coverage	experiment	map
2018	Directed test generation using concolic testing on RTL models	rtl	concolic	test-coverage	experiment	deep
2018	End-to-End Automated Exploit Generation for Validating the Security of Processor Designs	rtl	classical	security	experiment	deep
2018	Scalable Hardware Trojan Activation by Interleaving Concrete Simulation and Symbolic Execution	rtl	concolic	security	experiment	map
2018	Symbolic execution based test-patterns generation algorithm for hardware Trojan detection	rtl	classical	security	experiment	map
2019	Automated Activation of Multiple Targets in RTL Models using Concolic Testing	rtl	concolic	test-coverage	experiment	map
2020	Hardware/Software Co-verification Using Path-based Symbolic Execution	mixed-level	classical	functional	experiment	deep
2020	Selective Concolic Testing for Hardware Trojan Detection in Behavioral SystemC Designs	systemc-tlm	concolic	security	experiment	map
2021	Directed Test Generation for Activation of Security Assertions in RTL Models	rtl	concolic	security	experiment	map
2021	FuCE: Fuzzing+Concolic Execution guided Trojan Detection in Synthesizable Hardware Designs	rtl	selective-hybrid	security	experiment	map
2021	Scalable Concolic Testing of RTL Models	rtl	concolic	test-coverage	experiment	deep
2021	SoCCAR: Detecting System-on-Chip Security Violations Under Asynchronous Resets	rtl	concolic	security	experiment	map
2022	Achieving high coverage in hardware equivalence checking via concolic verification	mixed-level	concolic	equivalence	experiment	map
2022	EISec: Exhaustive Information Flow Security of Hardware Intellectual Property Utilizing Symbolic Execution	gate-netlist	classical	security	experiment	deep
2023	Processor Verification using Symbolic Execution: A RISC-V Case-Study	mixed-level	classical	equivalence	case-study	deep
2023	SEIF: Augmented Symbolic Execution for Information Flow in Hardware Designs	rtl	classical	security	experiment	deep
2023	Sylvia: Countering the Path Explosion Problem in the Symbolic Execution of Hardware Designs	rtl	classical	functional	experiment	deep
2024	Incremental Concolic Testing of Register-Transfer Level Designs	rtl	concolic	test-coverage	experiment	map
2024	Using Symbolic Execution to Analyze the Hardware TCP Protocol	hls	classical	test-coverage	case-study	deep
2025	FuSS: Coverage-Directed Hardware Fuzzing with Selective Symbolic Execution	rtl	selective-hybrid	test-coverage	experiment	deep
2026	AutoVeriFix+: High-Correctness RTL Generation via Trace-Aware Causal Fix and Semantic Redundancy Pruning	rtl	concolic	functional	experiment	deep
2026	Comparing Methods for the Cross-Level Verification of SystemC Peripherals With Symbolic Execution	mixed-level	classical	equivalence	experiment	deep

The canonical rows, rationales, URLs, and citation-chase provenance remain in the public survey record; this rendered table is a direct view, not a second hand-maintained data source.

13. Conclusion

Symbolic execution of digital hardware designs is a coherent survey scope when the name is tied to an operational test. The design or documented derived representation must run with symbolic hardware values; alternatives distinguished by that design representation over control and, for sequential designs, time must have predicates; feasibility must affect which execution is constructed next; and that mechanism must produce the paper's verification evidence. A derived or HLS representation also requires a semantic bridge adequate to the design-level claim. This boundary includes classical, concolic, and selective-hybrid execution. It excludes symbolic simulation, STE, BMC, trace-only search, generic synthesizable-source analysis, and forks confined to an external testbench unless they meet the same design-path conditions.

The bounded map contains 31 full-text-qualified publication records, including preprints and agent-adjudicated records, under a non-closed protocol with 1 failed query and no item-level human screening. That slice supports a focused account, with RTL as the principal design target and smaller SystemC, mixed-level, netlist, other-HDL, and HLS edges. It does not estimate the size or maturity of a field. Across the deep reads, difficult replayable witnesses are a recurring positive outcome.

Hardware changes the technique at its semantic core. Clocking and process composition, plus scheduling where it is semantically real, reset, translation, and environment determine path identity and claim strength. Guidance, backward search, fragments, caching, time abstraction, and fuzzing handoffs make useful searches practical by reducing selected work, reusing results, or omitting distinctions. Their end-to-end effects require accounting across the executor, formulas, concrete frontier, and semantic bridge; no cost quantity is conserved.

The resulting discipline is simple. State the executed artifact, initial state, time model, environment, exactness, returned evidence, and completion. Report positive witnesses separately from bounded exhaustion and incomplete search. With semantic conformance, shared benchmarks, outcome partitions, and effort measurements, this bounded technical slice can become much easier to evaluate and reuse without losing the narrow definition that makes it intelligible.

References

Ahmed, A., & Mishra, P. (2017). QUEBS: Qualifying Event Based Search in Concolic Testing for Validation of RTL Models. *2017 IEEE International Conference on Computer Design (ICCD)*, 185–192. https://doi.org/10.1109/iccd.2017.36

Ahmed, A., Farahmandi, F., & Mishra, P. (2018). Directed Test Generation Using Concolic Testing on RTL Models. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1538–1543. https://doi.org/10.23919/DATE.2018.8342260

Ahmed, A., Farahmandi, F., Iskander, Y., & Mishra, P. (2018). Scalable Hardware Trojan Activation by Interleaving Concrete Simulation and Symbolic Execution. *IEEE International Test Conference (ITC)*, 1–10. https://doi.org/10.1109/TEST.2018.8624854

Bagri, S. (2015). *Improving Branch Coverage in RTL Circuits with Signal Domain Analysis and Restrictive Symbolic Execution* [Master's thesis]. https://vtechworks.lib.vt.edu/server/api/core/bitstreams/d55b7807-e260-4b56-9cfd-3d0bbd4973bf/content

Baldoni, R., Coppa, E., D'Elia, D. C., Demetrescu, C., & Finocchi, I. (2018). A Survey of Symbolic Execution Techniques. *ACM Computing Surveys*, *51*(3), 1–39. https://doi.org/10.1145/3182657

Bruns, N., Herdt, V., & Drechsler, R. (2023). Processor Verification using Symbolic Execution: A RISC-V Case-Study. *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1–6. https://doi.org/10.23919/date56975.2023.10137202

Camurati, P., & Prinetto, P. (1988). Formal verification of hardware correctness: introduction and survey of current research. *Computer*, *21*(7), 8–19. https://doi.org/10.1109/2.65

Carter, W., Joyner, W., & Brand, D. (1979). Symbolic Simulation for Correct Machine Design. *16th Design Automation Conference*, 280–286. https://doi.org/10.1109/dac.1979.1600119

Debnath, M., Chowdhury, A. B., Saha, D., & Sur-Kolay, S. (2021). FuCE: Fuzzing+Concolic Execution Guided Trojan Detection in Synthesizable Hardware Designs. *arXiv*. https://doi.org/10.48550/arXiv.2111.00805

Debnath, M., Chowdhury, A. B., Saha, D., & Sur-Kolay, S. (2022). GreyConE: Greybox Fuzzing + Concolic Execution Guided Test Generation for High Level Designs. *2022 IEEE International Test Conference (ITC)*, 494–498. https://doi.org/10.1109/itc50671.2022.00059

Feng, T., Wang, L.-C., Cheng, K.-T., & Lin, A. C.-C. (2004). Improved Symbolic Simulation by Dynamic Functional Space Partitioning. *Design, Automation and Test in Europe Conference and Exhibition*, 42–47. https://doi.org/10.1109/DATE.2004.1268825

Fowze, F., Choudhury, M., & Forte, D. (2022). EISec: Exhaustive Information Flow Security of Hardware Intellectual Property Utilizing Symbolic Execution. *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 1–6. https://doi.org/10.1109/AsianHOST56390.2022.10022071

Harrath, N., Monsuez, B., & Delacroix, J. (2011). Building SystemC Waiting State Automata. *Proceedings of the Fifth International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2011)*, 1–12. https://doi.org/10.14236/ewic/VECOS2011.11

Hu, N. (2024). *Using Symbolic Execution to Analyze the Hardware TCP Protocol* [Master's thesis]. https://digitalcommons.unl.edu/computerscidiss/241/

Jayasena, A., & Mishra, P. (2024). Directed Test Generation for Hardware Validation: A Survey. *ACM Computing Surveys*, *56*(5), 1–36. https://doi.org/10.1145/3638046

Jayasena, A., Nallapaneni, S. S., & Mishra, P. (2025). FuSS: Coverage-Directed Hardware Fuzzing with Selective Symbolic Execution. *ACM Transactions on Embedded Computing Systems*, *24*(5s), 1–24. https://doi.org/10.1145/3760529

Kölbl, A., Kukula, J., & Damiano, R. (2001). Symbolic RTL Simulation. *38th Design Automation Conference (DAC)*, 47–52. https://doi.org/10.1145/378239.378278

Lin, B., Chen, J., & Xie, F. (2020). Selective Concolic Testing for Hardware Trojan Detection in Behavioral SystemC Designs. *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 19–24. https://doi.org/10.23919/date48585.2020.9116384

Lin, B., Cong, K., Yang, Z., Liao, Z., Zhan, T., Havlicek, C., & Xie, F. (2018). Concolic testing of SystemC designs. *2018 19th International Symposium on Quality Electronic Design (ISQED)*, 1–7. https://doi.org/10.1109/isqed.2018.8357256

Lin, B., Yang, Z., Cong, K., & Xie, F. (2016). Generating high coverage tests for SystemC designs using symbolic execution. *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 166–171. https://doi.org/10.1109/aspdac.2016.7428006

Lyu, Y., & Mishra, P. (2021). Scalable Concolic Testing of RTL Models. *IEEE Transactions on Computers*, *70*(7), 979–991. https://doi.org/10.1109/tc.2020.2997644

Lyu, Y., Ahmed, A., & Mishra, P. (2019). Automated Activation of Multiple Targets in RTL Models using Concolic Testing. *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 354–359. https://doi.org/10.23919/date.2019.8714989

Meng, X., Raj, K., Nath, A. P. D., Basu, K., & Ray, S. (2021). SoCCAR: Detecting System-on-Chip Security Violations Under Asynchronous Resets. *58th ACM/IEEE Design Automation Conference (DAC)*, 625–630. https://doi.org/10.1109/DAC18074.2021.9586291

Mukherjee, R., Joshi, S., O'Leary, J., Kroening, D., & Melham, T. (2020). Hardware/Software Co-verification Using Path-based Symbolic Execution. *arXiv*. https://doi.org/10.48550/arXiv.2001.01324

Mukherjee, R., Kroening, D., & Melham, T. (2015). Hardware Verification Using Software Analyzers. *2015 IEEE Computer Society Annual Symposium on VLSI*, 7–12. https://doi.org/10.1109/isvlsi.2015.107

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, *64*, 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

Pinto, S. (2017). *RTL Functional Test Generation Using Factored Concolic Execution* [Master's thesis]. http://hdl.handle.net/10919/78397

Qin, X., & Mishra, P. (2014). Scalable Test Generation by Interleaving Concrete and Symbolic Execution. *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, 104–109. https://doi.org/10.1109/vlsid.2014.25

Roy, P., & Chaki, S. (2022). Achieving high coverage in hardware equivalence checking via concolic verification. *Formal Methods in System Design*, *60*(3), 329–349. https://doi.org/10.1007/s10703-023-00414-1

Rudkowski, K. A., Ahmadi-Pour, S., & Drechsler, R. (2026). Comparing Methods for the Cross-Level Verification of SystemC Peripherals With Symbolic Execution. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *45*(8), 3794–3807. https://doi.org/10.1109/tcad.2025.3641038

Ryan, K., & Sturton, C. (2023). Sylvia: Countering the Path Explosion Problem in the Symbolic Execution of Hardware Designs. *Proceedings of the 23rd Conference on Formal Methods in Computer-Aided Design (FMCAD 2023)*, 110–121. https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_19

Ryan, K., Gregoire, M., & Sturton, C. (2023). SEIF: Augmented Symbolic Execution for Information Flow in Hardware Designs. *Hardware and Architectural Support for Security and Privacy (HASP)*, 1–9. https://doi.org/10.1145/3623652.3623666

Tan, Y., Meng, X., Jiang, Z., & Lyu, Y. (2026). AutoVeriFix+: High-Correctness RTL Generation via Trace-Aware Causal Fix and Semantic Redundancy Pruning. *arXiv*. https://doi.org/10.48550/arXiv.2603.11489

Witharana, H., Jayasena, A., & Mishra, P. (2024). Incremental Concolic Testing of Register-Transfer Level Designs. *ACM Transactions on Design Automation of Electronic Systems*, *29*(3), 1–23. https://doi.org/10.1145/3655621

Witharana, H., Lyu, Y., & Mishra, P. (2021). Directed Test Generation for Activation of Security Assertions in RTL Models. *ACM Transactions on Design Automation of Electronic Systems*, *26*(4), 1–28. https://doi.org/10.1145/3441297

Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. *18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 1–10. https://doi.org/10.1145/2601248.2601268

Yang, Z., Che, W., Zheng, Z., Hu, G., & Zhang, H. (2026). Forbench: Symbolic Simulation Helps Make Your Testbench More Formal. *arXiv*. https://doi.org/10.48550/arXiv.2608.01045

Zhang, R., Deutschbein, C., Huang, P., & Sturton, C. (2018). End-to-End Automated Exploit Generation for Validating the Security of Processor Designs. *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 815–827. https://doi.org/10.1109/micro.2018.00071

Zhang, Y., Feng, W., & Huang, M. (2016). Automatic generation of high-coverage tests for RTL designs using software techniques and tools. *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, 856–861. https://doi.org/10.1109/iciea.2016.7603701