



GIOLIT | FORMAL VERIFICATION OF SYSTEMVERILOG DESIGNS

Formal Verification Report: Assertion Failures

Document GIO-SVA-785AAA26C149 | issued 2026-10-14 16:21:16 UTC (UTC+0000)

PROFESSIONAL LICENSE No. GIOLIT-PRO-2026-000001 licensed to **Giolit Labs****counter_bug violates 1 assertion.**

A concrete run of the design was found for each:

✗ p_range (line 13) fails in clock cycle 11

Each failing run is shown cycle by cycle and as a waveform on page 3, and comes with a replay testbench for your simulator.

Design	top module counter_bug, 1 property
File	02_counter_bug.sv (14 lines) SHA-256 7e62e1a20bc7bd13efa13b805779672905de19b00dd2ffa8dc7b9dd9a475835
Reset	rst active in the first clock cycle
Issued to	Giolit Labs
Machine ID	5186e2bbb5474f2bb16093c28c3e092a
Machine	95bb1e6c01c1 — Ubuntu 24.04.5 LTS (kernel 7.2.9-orbstack-00391-g2b298d05fbd4); VirtualApple @ 2.50GHz
Verified by	95bb1e6c01c1
Engine	Giolit SystemVerilog Assertion Verifier 1.0.0, build 29f9c97
Date and time	2026-10-14 16:21:16 UTC (UTC+0000) (analysis 0.3 s)
Technique	Exhaustive state exploration
Resilience	not rated: no numeric bound in the proved assertions
License	Professional license No. GIOLIT-PRO-2026-000001 , licensed to Giolit Labs

How your client can check this document

- 1. Same design.** sha256sum 02_counter_bug.sv must print 7e62e1a20bc7bd13efa13b805779672905de19b00dd2ffa8dc7b9dd9a475835.
- 2. Licensed.** Issued under Giolit professional license No. GIOLIT-PRO-2026-000001, licensed to **Giolit Labs**.
- 3. Same result.** reproduce.sh re-runs the verification; every counterexample replays in a simulator (evidence/).
- 4. Ask us.** Write to business.giolitlabs@gmail.com quoting document GIO-SVA-785AAA26C149.

**Giolit — Formal Verification Services**

Mathematical proof that your RTL design meets its SystemVerilog assertions.

We offer formal verification as a service: proofs of the SystemVerilog assertions of RTL designs (protocols, data integrity, arbitration, liveness), formal sign-off evidence for safety standards (ISO 26262, DO-254, IEC 61508), assertion writing, and review of counterexamples with your engineers.

www.giolit.com business.giolitlabs@gmail.com

www.giolit.com

Properties and results

#	Property	Kind	Line	Result	Method or reason
1	counter_bug.p_range	assert	13	✖ failed	Exhaustive state exploration



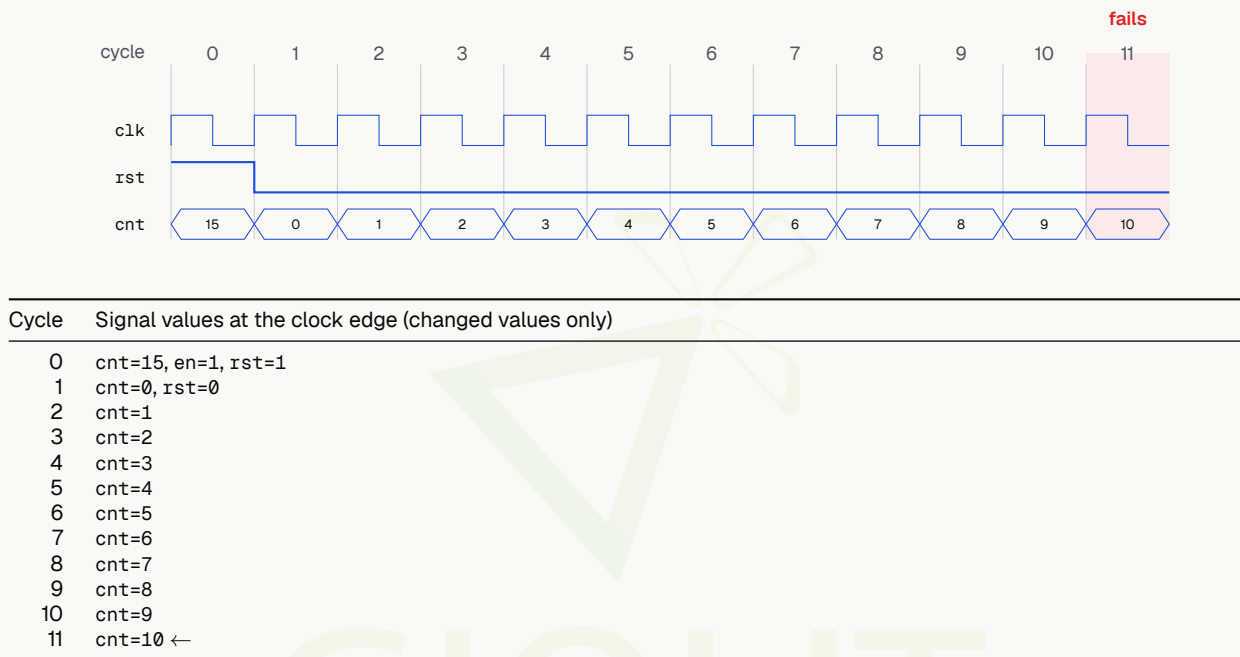
Counterexamples and witnesses

Each run starts at clock cycle 0 (the first clock edge) and is shown as a timing diagram of the signals that change, then cycle by cycle with the values that change. Registers without a reset start with the values shown in cycle 0. The run is also in the bundle as a waveform (VCD, for GTKWave or any waveform viewer) and as a replay testbench that drives the same inputs into the design in your simulator.

counter_bug.p_range (line 13, ✖ failed)

p_range: assert property (@(posedge clk) disable iff (rst) cnt <= 4'd9);

Found by: Exhaustive state exploration. 12 clock cycles; the property fails in cycle 11.



Evidence: evidence/counter_bug.p_range.vcd, evidence/counter_bug.p_range_replay.sv.

Safety margins and the Giolit Safety Resilience Score

A proof says the assertion holds; the **safety margin** says by how much. For every numeric bound of a proved assertion — a comparison such as `count <= DEPTH`, or the upper end of a delay window such as `##[1:3]` — the verifier tightened the bound step by step and proved the tightened assertion again. The margin is the largest tightening that still holds: the distance between the values (or latencies) the design actually reaches, over all runs, and the closest value the assertion forbids. It is exact when tightening one step further fails with a counterexample; otherwise it is a lower bound ($\geq$). A margin of 0 means the design reaches its bound — often by design, such as a FIFO that fills up — and that any change that moves the design toward the bound turns the verdict.

No proved assertion of this design has a numeric bound (all are exact relations such as equalities, one-hot conditions or orderings of events), so it has no safety margin to measure.

Semantics and scope

- **Clock.** One step of the model is one clock cycle. Every clocked process is clocked by one global clock unless the run used `--multiclock` (exact edge semantics for several or derived clocks).

- **Inputs.** The top module's inputs are free in every cycle: the results cover *every* input sequence, constrained only by the design's assume properties.
- **Start state.** Registers without an initial value start with arbitrary values; the reset `rst` is active in the first cycle and free afterwards.
- **Values.** Two-state semantics: X and Z values are arbitrary 0/1 values; asynchronous resets act at the clock edge. Several drivers of a net that can release it (z) are checked for contention.
- **Assertions.** Concurrent assertions follow IEEE 1800-2017 (sampled values, `disable iff, $past`); liveness properties are proved by k-liveness or refuted by an infinite run (lasso).

The design

02_counter_bug.sv

SHA-256 7e62e1a20bc7bd13efa13b805779672905de19b00dd2ffaf8dc7b9dd9a475835

```
1 // The same counter with a seeded bug: it wraps at 10 instead of 9.
2 // Expected (sva-verify --reset=rst): p_range FAILED with a 12-cycle counterexample.
3 module counter_bug (
4     input logic    clk,
5     input logic    rst,
6     input logic    en,
7     output logic [3:0] cnt
8 );
9     always_ff @(posedge clk)
10         if (rst) cnt <= 4'd0;
11         else if (en) cnt <= (cnt == 4'd10) ? 4'd0 : cnt + 4'd1;
12
13     p_range: assert property (@(posedge clk) disable iff (rst) cnt <= 4'd9);
14 endmodule
```

Reproducing this result

The bundle delivered with this document contains:

- 02_counter_bug.sv — the verified design file, unchanged
- certificate.json — the statement of this result, with the SHA-256 of every design and evidence file
- proof.json — the proofs (relations over the design state, and as stated by the engine) and the runs
- results.json — per-property results (machine-readable)
- evidence/*.vcd — every counterexample and witness as a waveform
- evidence/*_replay.sv — ... and as a replay testbench for a SystemVerilog simulator
- verifier-output.txt — the verifier's console report
- reproduce.sh — the command that produced this result
- certificate-password.txt — the password that opens counterexample-report.pdf

Command (run in the bundle folder):

```
sva-verify --reset=rst 02_counter_bug.sv
```

About Giolit

We offer formal verification as a service: proofs of the SystemVerilog assertions of RTL designs (protocols, data integrity, arbitration, liveness), formal sign-off evidence for safety standards (ISO 26262, DO-254, IEC

61508), assertion writing, and review of counterexamples with your engineers. Visit www.giolit.com or write to business.giolitlabs@gmail.com.



Giolit — Formal Verification Services

Mathematical proof that your RTL design meets its SystemVerilog assertions.

We offer formal verification as a service: proofs of the SystemVerilog assertions of RTL designs (protocols, data integrity, arbitration, liveness), formal sign-off evidence for safety standards (ISO 26262, DO-254, IEC 61508), assertion writing, and review of counterexamples with your engineers.

 www.giolit.com

 business.giolitlabs@gmail.com



www.giolit.com