



GIOLIT | FORMAL VERIFICATION OF SYSTEMVERILOG DESIGNS

Certificate of Formal Verification

Document GIO-SVA-9564A1822018 | issued 2026-10-14 16:21:03 UTC (UTC+0000)

PROFESSIONAL LICENSE No. GIOLIT-PRO-2026-000001 licensed to **Giolit Labs**

Every assertion of margins holds.

For every input sequence, from every state after the reset `rst`, in every clock cycle:

✓ 2 assertions proved: `a_level`, `a_latency`

This is a **mathematical proof** covering all runs of the design, not a simulation of some of them. The proofs are part of this certificate (page 3).

Design	top module <code>margins</code> , 2 properties
File	<code>11_margins.sv</code> (40 lines)
	SHA-256 6965d03abffe468bcb29947e9a9efd71e21e7f159f7f6f2af357f7e9ef5446df
Reset	<code>rst</code> 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 16:21:03 UTC (UTC+0000) (analysis 0.4 s)
Technique	Exhaustive state exploration
Resilience	★★★★☆ Giolit Safety Resilience Score 3 of 5 (page 3)
License	Professional license No. GIOLIT-PRO-2026-000001 , licensed to Giolit Labs

How your client can check this document

1. **Same design.** `sha256sum 11_margins.sv` must print 6965d03abffe468bcb29947e9a9efd71e21e7f159f7f6f2af357f7e9ef5446df.
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-9564A1822018.



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	margins.a_level	assert	38	✓ proved	Exhaustive state exploration
2	margins.a_latency	assert	39	✓ proved	Exhaustive state exploration



The proofs

For each proved property: the property as written, the method that proved it, and the proof in terms of the design's signals, simplified (relations implied by the signal widths are omitted). The complete machine-readable proofs are in `proof.json`.

`margins.a_level` (line 38,)

```
a_level: assert property (level <= 4'd12);
```

Method: Exhaustive state exploration. Every reachable state of the design together with its property checkers was enumerated, for every input sequence; no reachable state violates the property.

The proof. Inv below is the exact set of states of the design (with its property checkers) reachable at a clock edge, as enumerated: every case is a reachable state or a range of them. It holds at the first clock edge, every cycle from a state in it leads to a state in it, and no cycle from a state in it violates the property.

Inv = the relations that hold in every case, and one of the cases:

in every case

`a_latency:age10 = 0 ∧`

`a_latency:age11 = 0 ∧`

`a_latency:age12 = 0 ∧`

`a_latency:s0 = 0`

case 1 `$initstate = 0 ∧ 0 ≤ level ≤ 9 ∧ a_latency:age7 = 0 ∧ a_latency:age8 = 0 ∧ a_latency:age9 = 1`
`∧ ack = 1 ∧ age = 1 ∧ busy = 0`

case 2 `$initstate = 0 ∧ 0 ≤ level ≤ 9 ∧ a_latency:age7 = 0 ∧ a_latency:age8 = 1 ∧ a_latency:age9 = 0`
`∧ ack = 0 ∧ age = 1 ∧ busy = 1`

case 3 `$initstate = 0 ∧ 0 ≤ level ≤ 9 ∧ a_latency:age7 = 1 ∧ a_latency:age8 = 0 ∧ a_latency:age9 = 0`
`∧ ack = 0 ∧ age = 0 ∧ busy = 1`

case 4 `$initstate = 1 ∧ 0 ≤ age ≤ 1 ∧ 0 ≤ level ≤ 9 ∧ a_latency:age7 = 0 ∧ a_latency:age8 = 0 ∧`
`a_latency:age9 = 0 ∧ ack = 1 ∧ busy = 0`

case 5 `$initstate = 1 ∧ 0 ≤ level ≤ 9 ∧ 2 ≤ age ≤ 3 ∧ a_latency:age7 = 0 ∧ a_latency:age8 = 0 ∧`
`a_latency:age9 = 0 ∧ busy = 0`

case 6 `$initstate = 1 ∧ 10 ≤ level ≤ 15 ∧ a_latency:age7 = 0 ∧ a_latency:age8 = 0 ∧ a_latency:age9 =`
`0`

case 7 `$initstate = 1 ∧ 0 ≤ level ≤ 9 ∧ a_latency:age7 = 0 ∧ a_latency:age8 = 0 ∧ a_latency:age9 = 0`
`∧ busy = 1`

case 8 `0 ≤ age ≤ 1 ∧ 0 ≤ level ≤ 9 ∧ a_latency:age7 = 0 ∧ a_latency:age8 = 0 ∧ a_latency:age9 = 0 ∧`
`ack = 0 ∧ busy = 0`

where `$initstate` is 1 in the first clock cycle only; `a_latency:age10`, `a_latency:age11`, `a_latency:age12`, `a_latency:age7`, `a_latency:age8`, `a_latency:age9` are counters of the property checkers (clock cycles since an attempt started); `a_latency:s0` is a state of the property checkers (`p:sK`: 1 while an attempt of property `p` is at stage `K` of its sequence). Every other name is a signal of the design, `x[k]` bit `k` of `x`.

`margins.a_latency` (line 39,)

```
a_latency: assert property (!busy && req |-> ##[1:6] ack);
```

Method: Exhaustive state exploration. **The proof** is the same invariant Inv as for `margins.a_level` above: no state in it violates this property either.

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.

Property	Bound	Margin	Allowed	Relative	Score
a_level	level <= 4'd12	3	13 values	23%	★★★★☆
a_latency	##[1:6]	3 cycles	6 cycles	50%	★★★★★

The score. Each bound is scored by its relative margin (margin / size of the allowed range): ★★★★★ from 50%, ★★★★★☆ from 25%, ★★★★★☆ from 10%, ★★★★★ above 0, ★☆☆☆☆ when the design reaches the bound. The design's **Giolit Safety Resilience Score** is the score of its weakest bound: ★★★★★ (3 of 5).

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

11_margins.sv

SHA-256 6965d03abffe468bcb29947e9a9efd71e21e7f159f7f6f2af357f7e9ef5446df

```

1 // Safety margins: a buffer that never holds more than 9 entries, checked against a limit of 12,
  and a responder that
2 // answers in exactly 3 cycles, checked against a window of 1 to 6 cycles.
3 // Expected (sva-verify --reset=rst --margins 11_margins.sv): both PROVED; a_level has margin 3 of
  13 values, a_latency
4 // margin 3 of 6 cycles.
5 module margins (
6     input logic    clk,
7     input logic    rst,
8     input logic    put,
9     input logic    take,
10    input logic    req,
11    output logic [3:0] level,
12    output logic    ack
13 );
14 // the buffer: put adds an entry while fewer than 9 are held, take removes one
15 always_ff @(posedge clk)
16     if (rst) level <= '0;
17     else if (put && !take && level < 4'd9) level <= level + 4'd1;
18     else if (take && !put && level != 4'd0) level <= level - 4'd1;
19
20 // the responder: ack exactly 3 cycles after a request it accepts
21 logic [1:0] age;
22 logic      busy;
```

```
23 always_ff @(posedge clk)
24   if (rst) begin
25     busy <= 1'b0; ack <= 1'b0; age <= '0;
26   end else begin
27     ack <= 1'b0;
28     if (!busy && req) begin busy <= 1'b1; age <= '0; end
29     else if (busy) begin
30       if (age == 2'd1) begin ack <= 1'b1; busy <= 1'b0; end
31       else age <= age + 2'd1;
32     end
33   end
34
35 default clocking cb @(posedge clk); endclocking
36 default disable iff (rst);
37
38 a_level:  assert property (level <= 4'd12);
39 a_latency: assert property (!busy && req |-> ##[1:6] ack);
40 endmodule
```

Reproducing this result

The bundle delivered with this document contains:

- 11_margins.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)
- verifier-output.txt — the verifier's console report
- reproduce.sh — the command that produced this result
- certificate-password.txt — the password that opens certificate.pdf

Command (run in the bundle folder):

```
sva-verify --reset=rst --margins 11_margins.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