



AI Requirements Analysis

Overview

Traditional verification methods catch coding errors, however they miss the most critical issue: requirements that are never properly implemented. The Visual Verification™ Suite - AI Requirements Analysis tool understands design intent and verifies the implementation at the semantic level.

Highlights

- **Semantic Requirement Matching:** Automatically verifies if HDL code implements specified requirements, even when variable names and comments differ from requirement text
- **Real-Time IDE Integration:** The Visual Verification Suite AI Requirements extension provides instant feedback as you code - see requirement compliance status without leaving your development environment
- **Secure On-Premises Deployment:** Your code never leaves your infrastructure. Run on-premises using open-weight models for complete data sovereignty. Suggested hardware Nvidia RTX Pro 6000 Blackwell Workstation
- **FPGA Specialized Analysis:** Understands HDL-specific patterns: clock domain crossings, metastability protection, timing constraints, and resource utilization
- **Continuous Verification:** Integrates with CI/CD pipelines. Automatically re-verifies requirements as code changes, catching regressions immediately

How It works

1. **Define Requirements:** Specify structured requirements in plain English using a simple CSV spreadsheet
2. **Develop HDL Code:** Write VHDL, Verilog, or SystemVerilog as usual.
3. **Automatic Analysis:** 480 billion parameter “thinking” model analyzes your code against requirements, identifying implementations and detecting gaps or contradictions
4. **Review Results:** See compliance status in real-time, with highlighted code sections showing where requirements are met

The screenshot shows the 'Text Reports' application window. On the left is a navigation pane with a tree view containing various report categories like 'Design Summary', 'Clocks', 'Gated Clocks', etc. The main area is split into two panes. The top pane displays a table of verification results with columns for ID, Status, Confidence, Title, Feedback, File, Start Line, End Line, and Reason. The bottom pane shows a code editor with Verilog code, where lines 9 and 10 are highlighted in yellow.

ID	Status	Confidence	Title	Feedback	File	Start Line	End Line	Reason
1 REQ-001	PASS	HIGH	Synchronous Reset Initialization	The implementation satisfies the requirement: "The counter register cntff SHALL be asynchronously reset to the value 4'b0000 on the falling edge of the active-low reset signal rst, independently of the clock, ensuring a defined initial state prior to any counting operation.". All mandatory control paths were identified.	counter_gate_deco...	5	8	Primary implementation site for this requirement.
2 REQ-001	PASS	HIGH	Synchronous Reset Initialization	The implementation satisfies the requirement: "The counter register cntff SHALL be asynchronously reset to the value 4'b0000 on the falling edge of the active-low reset signal rst, independently of the clock, ensuring a defined initial state prior to any counting operation.". All mandatory control paths were identified.	counter_gate_deco...	12	15	Secondary implementation site for this requirement.
3 REQ-002	PASS	HIGH	Mutually Exclusive Decode Output Assertion	The implementation satisfies the requirement: "The decoded outputs e1 and e2 SHALL NOT be simultaneously asserted. e1 SHALL be asserted (logic 1) only when all bits of cntff are zero (count = 0), and e2 SHALL be asserted (logic 1) only when all equals 4'b0111 (count = 7); all other counter states SHALL produce both outputs deasserted.". All mandatory control paths were identified.	simple_fp.v	3	8	Primary implementation site for this requirement.

```

1 module simple_fp (din1, din2, f1, clk, rst, out);
2   input din1, din2, f1, clk, rst;
3   output out;
4   reg q1, out;
5   wire e1, e2, mx1, mx2;
6
7   counter_counter_inst(.clk(clk), .rst(rst), .e1(e1), .e2(e2));
8
9   assign mx1 = (e1) ? q1 : din1;
10  assign mx2 = (e2) ? mx1 : din2;
11
12  always @ (posedge clk)
13  -begin
14    q1 <= f1;
15  end
16  always @ (posedge clk)
17    out <= mx2;
18  endmodule

```

User Benefits

- Catches functional issues and missing requirements early in the verification cycle
- Spotlights areas of code which might have functional issues
 - Saves time in simulation
 - Saves time finding potential root cause of issues
- Helps with third party assessor for safety critical applications e.g. ESA / DO-254 etc.
- Helps understand maturity of design
- Code can be generated by engineering team or AI Agentic Code generation

This product is currently in development with delivery scheduled for 2026