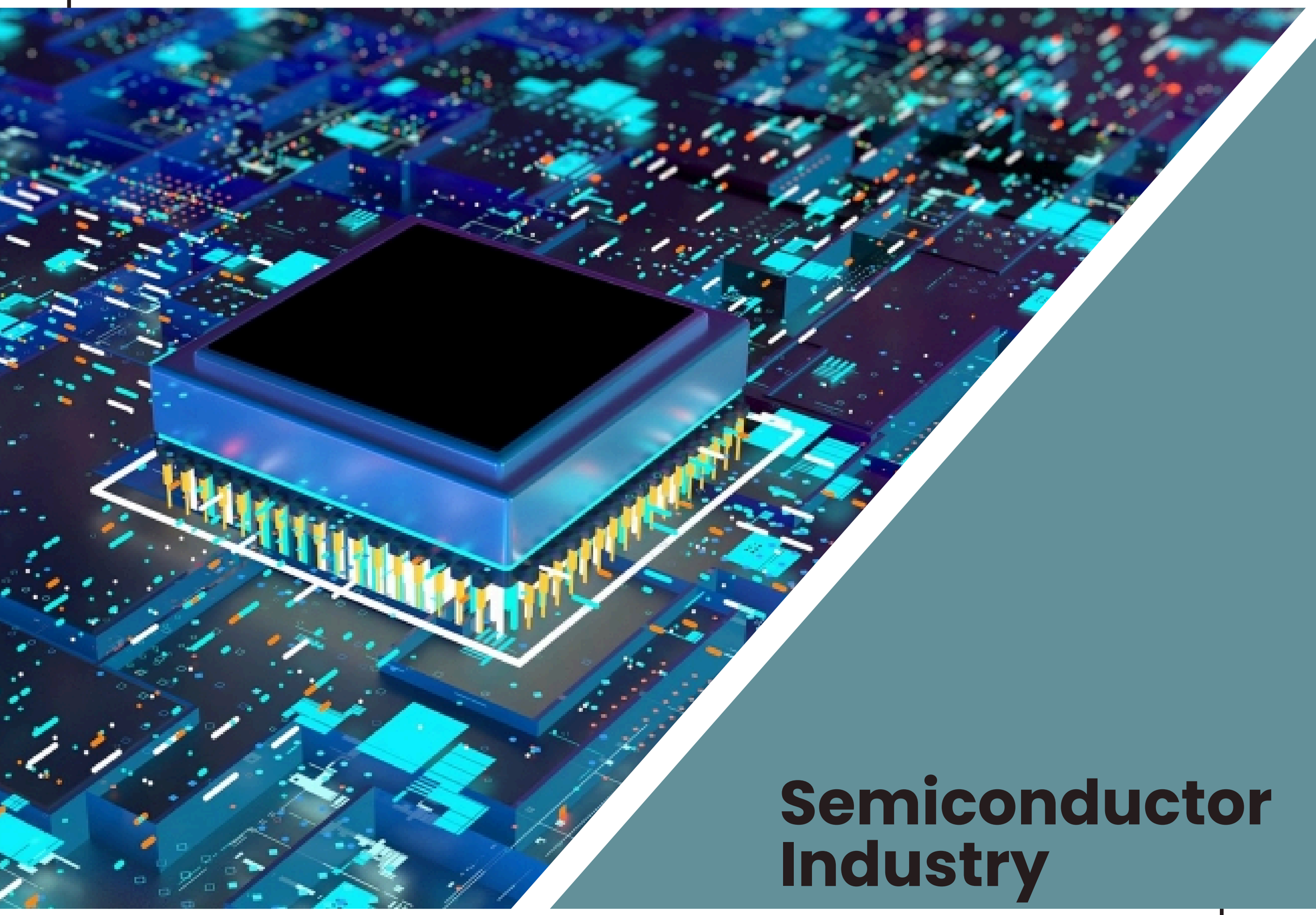
RTL TO GDS

Article-26

Design Verification Vs Formal Verification



Semiconductor Industry

Written By-

S. Chinna Venkata Narayana Reddy

Contents

- What is Formal Verification?
- What is Functional Verification(Design Verification)?
- What are the key differences?
- What is their significance in VLSI frontend verification?
- What are the key challenges ?

Design Verification Vs Formal Verification

Introduction:-

Functional Verification(Design Verification) and Formal Verification are two essential methods used in VLSI design to ensure the correctness of a digital circuit. Functional Verification relies on simulation to test the design against various input scenarios, while Formal Verification uses mathematical proofs to verify that the design meets its specifications under all conditions. Functional verification is more flexible, testing diverse scenarios, but formal verification offers exhaustive, corner-case analysis. Both approaches complement each other, providing a comprehensive verification solution. Together, they help improve the reliability and accuracy of complex VLSI systems.

Functional Verification:-

Functional Verification in the VLSI front-end design process is one of the most critical steps to ensure that a chip design behaves as expected before it goes through synthesis and eventually fabrication. It refers to the process of verifying that the digital logic design meets its functional specifications. Functional verification simulates the design in various scenarios to catch any discrepancies between the intended and actual behavior of the design.

Importance of Functional Verification:

- In the VLSI front-end, the design complexity is immense due to the integration of millions or even billions of transistors into a single chip. As the scale of designs grows, traditional manual testing is insufficient to ensure correctness. Functional verification becomes crucial because it allows the designer to ensure that every function of the system operates correctly under all conditions, without manufacturing the chip first.
- Errors discovered late in the design process, particularly after manufacturing, can be extremely expensive to fix. Hence, functional verification is employed early and often, enabling the design teams to detect and correct issues in the RTL code before the design goes to the synthesis stage.

Key Components of Functional Verification:

1. **Testbenches:** A testbench is an environment used to apply stimuli to the RTL design and monitor its output. It acts as the "driver" of the verification process, controlling inputs and verifying outputs. Modern testbenches are written using hardware description languages like Verilog, VHDL, or SystemVerilog, and they include all components necessary for running functional tests.
2. **Assertions:** Assertions are properties defined in the testbench that must always hold true during the simulation. These help in detecting any violations or incorrect behavior of the design. They ensure that specific conditions are always met, such as signal integrity, correct handshakes in protocols, or compliance with timing specifications.
3. **Test Vectors:** Test vectors are input patterns applied to the design to check its response. These vectors are generated based on functional requirements and simulate different scenarios, such as normal operating conditions, edge cases, and even unexpected or extreme conditions.
4. **Simulations:** Functional verification is primarily conducted using simulation, where the design is tested over time by running through different scenarios. Simulation tools mimic the behavior of the hardware design by executing the RTL code as if it were hardware. Functional

coverage metrics are used to determine how well the simulation exercises different parts of the design.

5. **Code Coverage and Functional Coverage:** Code coverage refers to the extent to which the RTL code has been executed during simulation. It helps ensure that all parts of the code have been tested. Functional coverage focuses on verifying that all functionalities and requirements have been tested. Both metrics are crucial in verifying that the design is comprehensively tested.
6. **Constrained Random Testing:** Instead of manually defining every test case, constrained random testing allows the testbench to generate random input stimuli within certain constraints. This method explores a broader range of scenarios, including unexpected and rare corner cases, to find bugs that might not be detected with manually written tests.
7. **Regression Testing:** This involves running a suite of verification tests (including those used in earlier design versions) whenever new changes are made to the design. This ensures that any new modification or optimization in the design doesn't break any previously working functionality.

Tools for Functional Verification:

There are several commercial and open-source tools available for functional verification, such as:

- **SystemVerilog/UVM** (Universal Verification Methodology): The most widely used methodology for functional verification, allowing for structured testbenches with features like constrained random testing, coverage-driven verification, and assertions.
- **ModelSim:** A widely used simulation tool that allows designers to run functional verification.
- **QuestaSim:** Another advanced simulation tool for verifying the functionality of complex VLSI designs.
- **VCS** (Verilog Compiler Simulator): A powerful tool from Synopsys, used for RTL simulation and verification.

The Role of Functional Verification in the VLSI Design Flow:

- Functional verification begins early in the design flow, right after the RTL code is developed. It is a continuous process that runs in parallel with design refinements. Once the RTL code is verified to meet all the functional specifications, the design proceeds to synthesis, place and route, and eventually to fabrication.
- At this stage, functional verification ensures that the logic and the intended functionality of the design are correct and meet the given specification. It also helps in detecting protocol violations, functional bugs, or design inefficiencies.
- In modern verification processes, functional verification involves not just simulation but also the integration of methodologies like coverage-driven verification (CDV) and assertion-based verification (ABV) to increase verification efficiency and confidence.

Benefits of Functional Verification:

- **Bug Detection at an Early Stage:** By simulating and testing various aspects of the design at the functional level, critical bugs are caught early, preventing costly fixes later in the design or post-silicon stages.
- **Improved Design Quality:** Continuous verification helps improve the robustness and reliability of the design, ensuring that it functions under different conditions without failure.

- **Reduced Risk and Time-to-Market:** Effective functional verification reduces the chance of errors slipping through to later stages, which minimizes the risk of silicon re-spins and delays in the production timeline.
- **Supports Iterative Development:** Since functional verification can be performed throughout the design process, it supports iterative development and allows changes to be verified quickly.

Challenges in Functional Verification:

- **Complexity:** Modern VLSI designs are highly complex, with millions of gates and intricate interconnections. Functional verification must simulate many different scenarios, which can be time-consuming and require significant computational resources.
- **Coverage:** It is difficult to guarantee 100% functional coverage since it's nearly impossible to test every possible scenario. Verification engineers must carefully choose the most critical cases to test.
- **Scalability:** As designs become larger, functional verification methodologies must scale accordingly, which poses challenges in managing verification at such a high level of complexity.

Formal Verification

Formal Verification is a rigorous method used in VLSI design to ensure that a design adheres to its specification. Unlike traditional simulation-based approaches, formal verification uses mathematical techniques to prove that a digital circuit behaves correctly under all possible input conditions. This makes formal verification a powerful tool in identifying corner-case bugs that might be missed during simulation. It has become increasingly important in modern chip design due to the growing complexity of integrated circuits (ICs) and the need for higher confidence in the correctness of designs.

Importance of Formal Verification:

- With the complexity of VLSI systems continuing to grow—encompassing billions of transistors, multiple functional units, and intricate protocols—traditional verification techniques, such as simulation, are often inadequate. Functional verification via simulation only checks the design for specific test cases, meaning that not all input scenarios can be covered. As designs become more complex, the likelihood of missing bugs through simulation increases, especially those that occur only under rare or unusual conditions.
- Formal verification fills this gap by mathematically proving that the design satisfies its specification for all possible inputs, not just a subset of inputs that simulation-based approaches can cover. This makes formal verification highly useful for ensuring the correctness of critical parts of a design, such as control logic, protocols, and interfaces, where small errors can lead to catastrophic system failures.

Key Concepts in Formal Verification:

1. **Mathematical Proofs:** Formal verification is based on the idea of creating a formal mathematical model of both the design and its specification. Verification tools use this model to prove, mathematically, that the design satisfies the specification. If the design does not meet the specification, the tools can provide a counterexample that shows where and how the design fails.

2. **Exhaustive Search:** One of the key strengths of formal verification is its exhaustive nature. Unlike simulation, which tests the design for specific input scenarios, formal verification checks the entire design for all possible inputs. This exhaustive search makes formal verification ideal for finding corner cases—rare, complex input sequences that may not be discovered using traditional testing methods.
3. **Properties:** Formal verification requires the specification of "properties" or "assertions," which are essentially conditions or behaviors that the design must meet. These properties are expressed in a formal language, such as SystemVerilog Assertions (SVA) or Property Specification Language (PSL). The formal verification tool then proves whether the design satisfies these properties.
4. **Model Checking:** Model checking is one of the primary techniques used in formal verification. In this approach, a finite-state model of the design is created, and the tool systematically checks whether this model satisfies the specified properties. If a violation is found, the tool generates a counterexample that shows how the property can be violated.
5. **Equivalence Checking:** Equivalence checking is a form of formal verification used to compare two versions of a design—typically the RTL and the synthesized gate-level netlist. The goal is to prove that the two representations are functionally equivalent, meaning that they produce the same outputs for all possible inputs. Equivalence checking ensures that no functional errors were introduced during the synthesis process.
6. **Theorem Proving:** Theorem proving is another formal verification technique that involves creating mathematical proofs for complex properties. Theorem provers allow for the verification of complex systems by breaking down large problems into smaller, more manageable sub-problems. While theorem proving is highly powerful, it can be more difficult to automate compared to model checking.

Applications of Formal Verification:

Formal verification is most commonly used in the following areas of VLSI design:

- **Control Logic:** Control logic often has many different states and transitions between these states, making it difficult to verify using simulation alone. Formal verification can ensure that the control logic behaves correctly for all possible sequences of inputs and states.
- **Protocol Compliance:** Many designs involve complex communication protocols, such as PCIe, USB, or AXI, that have strict timing and ordering requirements. Formal verification can be used to prove that the design adheres to the protocol specification under all conditions, ensuring that no protocol violations occur.
- **Safety-Critical Systems:** In safety-critical applications, such as automotive, medical devices, and aerospace, even small design errors can lead to catastrophic failures. Formal verification provides a higher level of assurance that the design is error-free, making it an essential part of the verification process for these systems.
- **Power Management Circuits:** Modern chips often include sophisticated power management circuitry, which must switch between different power states without violating timing or functional constraints. Formal verification can prove that these transitions are handled correctly.
- **Clock Domain Crossing (CDC):** Designs often include multiple clock domains, and incorrect handling of signals crossing between these domains can lead to metastability and

functional errors. Formal verification is used to ensure that all clock domain crossings are safe and that no data corruption occurs.

Benefits of Formal Verification:

- **Exhaustiveness:** Unlike simulation, which is limited to specific test cases, formal verification checks all possible scenarios. This exhaustiveness ensures that even rare corner-case bugs are detected and corrected, which improves the overall reliability of the design.
- **Higher Confidence in Design:** Formal verification provides a higher level of confidence in the correctness of a design. By mathematically proving that the design satisfies its specification, formal verification ensures that critical parts of the design are error-free. This is especially important in high-assurance applications, such as aerospace, defense, and automotive industries.
- **Early Bug Detection:** Formal verification can be applied early in the design process, even before RTL code is fully complete. By catching bugs early, formal verification reduces the risk of late-stage design changes, which are often costly and time-consuming.
- **Complementary to Simulation:** Formal verification complements simulation-based functional verification by providing coverage in areas where simulation is weak. While simulation tests a design for specific scenarios, formal verification ensures correctness for all possible inputs. When used together, simulation and formal verification provide a comprehensive verification solution.

Challenges of Formal Verification:

- **Complexity:** Formal verification can be computationally expensive, especially for large designs with many states and transitions. The complexity of the mathematical models used in formal verification can make the process time-consuming and resource-intensive.
- **Scalability:** As the size and complexity of modern IC designs grow, scaling formal verification to cover large designs remains a challenge. Verification tools need to handle designs with billions of gates, multiple clock domains, and intricate interactions, which can lead to performance bottlenecks.
- **Tool Expertise:** Formal verification tools often require a higher level of expertise compared to traditional simulation-based verification tools. Verification engineers must be well-versed in formal methods, including property specification languages and model checking techniques, to use these tools effectively
- **Limited Automation:** While model checking and equivalence checking are highly automated, more advanced techniques like theorem proving may require manual intervention. This can make formal verification more time-consuming compared to traditional verification methods.

Common Formal Verification Tools:

Several commercial and open-source tools are available for formal verification in VLSI design:

- **JasperGold (Cadence):** One of the most widely used formal verification tools, offering support for model checking, equivalence checking, and property verification.
- **VC Formal (Synopsys):** A comprehensive formal verification tool that provides equivalence checking, model checking, and formal property verification.
- **OneSpin (Siemens EDA):** Provides specialized formal verification solutions for control logic, safety-critical designs, and clock domain crossing.

Differences between Formal Verification and Functional Verification

Characteristic	Formal Verification	Functional Verification
Approach	Mathematical proofs	Simulations
Rigor	More rigorous	Less rigorous
Completeness	Can explore all possible states and transitions of the design	Cannot explore all possible states and transitions of the design
Flexibility	Less flexible	More flexible
Time and effort	More time-consuming and complex	Less time-consuming and less complex

Conclusion:-

In conclusion, both Functional and Formal Verification play critical roles in the VLSI verification process. Functional Verification offers flexibility and is ideal for testing various scenarios through simulation, while Formal Verification provides exhaustive mathematical proof for design correctness, ensuring coverage of all corner cases. Each approach has its strengths and limitations, but when used together, they create a robust and comprehensive verification strategy. The combination of these methods helps improve design quality, ensuring accuracy and reducing the risk of undetected errors.

THANK YOU !

The VLSI Voyager

chinna venkata narayana reddy seelam
cvnreddyseelam07@gmail.com