Chapter 6: RTL Verification using Simulation Methods
Simulation-Based Verification Techniques

6.1 Introduction to RTL Verification

Register Transfer Level (RTL) verification is a critical phase in the design of digital systems, where the functionality of the RTL code (usually written in Verilog or VHDL) is verified to ensure it behaves as expected. RTL verification helps identify design flaws early in the design process, reducing costly errors that might only be discovered during later stages or after fabrication.

Simulation-based verification plays a central role in this process. It allows designers to simulate the behavior of the hardware described by RTL code in a controlled environment and check if the design meets the specifications. This chapter covers the fundamental simulation-based verification techniques used in RTL verification, focusing on methods like functional simulation, timing simulation, and coverage-driven verification.

6.2 Types of Simulation in RTL Verification

6.2.1 Functional Simulation

Functional simulation is the most common form of simulation-based verification. It checks whether the design behaves as expected under a set of test inputs. During functional simulation, the simulator runs the RTL design with a set of predefined testbenches that stimulate the design and check its output.

- **Purpose: To verify that the design performs the intended functions.**
- **How it Works: The design is tested for logical correctness by applying inputs and checking outputs for expected results.**
- **Tools: Common tools for functional simulation include ModelSim, VCS (Verilog Compiler Simulator), XSIM, and Questa.**

Example: Functional Simulation Code (Verilog)

```
module testbench;

  reg clk, reset;
```

```
  wire [3:0] out;

  // Instantiate the design under test (DUT)
  my_design dut (
    .clk(clk),
    .reset(reset),
    .out(out)
  );

  // Clock generation
  always begin
    #5 clk = ~clk;  // 10 ns clock period
  end

  // Stimulus generation
  initial begin
    clk = 0;
    reset = 1;
    #10 reset = 0;  // Reset the DUT for 10 ns
    #20 reset = 1;  // Apply reset again at 20 ns
    #100 $finish;   // End simulation after 100 ns
  end
endmodule
```

This simple testbench stimulates the DUT (`my_design`) with a clock and reset signal, and the simulator will check if the outputs match the expected results.

6.2.2 Timing Simulation

Timing simulation is used to ensure that the RTL design meets the required timing constraints. In this type of simulation, the simulator takes into account propagation delays, setup and hold times, and clock skew, making it more realistic than functional simulation alone.

- **Purpose: To verify that the design will work correctly in the physical world under the constraints of time.**
- **How it Works: The simulator uses detailed timing models of the cells and interconnects to simulate the actual behavior of the circuit with respect to timing.**
- **Tools: Tools like PrimeTime, ModelSim, and VCS support timing simulations.**

Example: Timing Simulation Use Case

In a timing simulation, the simulator checks if the signals at each clock edge meet timing requirements, such as whether setup and hold times are satisfied for flip-flops.

6.2.3 Gate-Level Simulation

After the RTL code is synthesized into gate-level netlist (the design is transformed into a representation of logic gates), gate-level simulation is used to ensure that the synthesized design behaves as expected. This type of simulation is typically done after synthesis and before physical design.

- **Purpose: To check the logical correctness of the design after synthesis and to verify the impact of the synthesis process on the behavior.**
- **How it Works: The gate-level netlist is simulated in a manner similar to functional simulation, but it uses the synthesized netlist (with logic gates) instead of RTL code.**
- **Tools: Gate-level simulation is often performed with the same tools used for functional simulation (e.g., VCS, ModelSim), but with a synthesized netlist.**

6.3 Verification Techniques in RTL Simulation

6.3.1 Testbenches

A testbench is a specialized simulation environment used to apply inputs to the design and verify its outputs. The testbench includes:

- **Stimulus generation: Applying test inputs to the design.**
- **Monitor: Observing and reporting output values from the DUT (Design Under Test).**
- **Checker: Comparing the actual output with the expected output.**

There are two main types of testbenches in RTL verification:

Directed Testbenches: These use a predefined set of test vectors to verify specific scenarios. Directed tests are easy to write but may not catch all corner cases.

```
 initial begin
  // Test specific conditions
  reset = 1;
  #10 reset = 0;
  #20 reset = 1;
  // Apply more stimuli
end
```

-

Random Testbenches: These generate random input sequences to stress the design and test it under various conditions. Random testbenches are more exhaustive but harder to control.

```
 initial begin
  // Random stimuli generation using $random
  input_a = $random;
  input_b = $random;
end
```

-

6.3.2 Assertion-Based Verification

Assertions are used to specify properties that should hold true during simulation. Assertions help to automatically check the correctness of the design and are often used in combination with formal verification to prove that the design satisfies the specification.

- **Property Specification: Assertions allow the specification of properties like "if A happens, B must happen next" or "the output should never be high unless the input is high."**

Example: Assertion in Verilog

```
assert property (@(posedge clk) reset == 0);
```

This assertion ensures that `reset` is not high on the rising edge of `clk`.

6.3.3 Code Coverage and Functional Coverage

Code Coverage is used to measure how much of the RTL code has been exercised during simulation. It helps identify untested areas of the design.

- **Types of Coverage:**
    - **Statement Coverage: Ensures that each line of code is executed at least once.**
    - **Branch Coverage: Ensures that each possible branch in the code (e.g., `if-else` statements) is exercised.**
    - **Toggle Coverage: Measures how often signals change from 0 to 1 and vice versa.**

Functional Coverage is used to track whether all functional scenarios (e.g., all possible input combinations) have been tested. Functional coverage can be manually defined in the testbench or automatically tracked by the simulator.

Example: Functional Coverage in Verilog

```
covergroup cg;
```

```
  coverpoint signal_a;

  coverpoint signal_b;

endgroup
```

This coverage block tracks the coverage of `signal_a` and `signal_b`.

6.4 Verification Methodologies

6.4.1 UVM (Universal Verification Methodology)

The Universal Verification Methodology (UVM) is a widely used methodology in RTL verification. It standardizes testbenches to improve reusability and scalability. UVM provides:

- **Reusable testbenches**
- **Randomization**
- **Automatic checking through assertions**
- **Transaction-level communication between different components of the testbench**

Example: UVM Testbench Structure

```
class my_test extends uvm_test;

  `uvm_component_utils(my_test)


  function new(string name="my_test");

    super.new(name);

  endfunction


  virtual function void run_phase(uvm_phase phase);
```

```
    // Test logic
  endfunction
endclass
```

6.4.2 Formal Verification

Formal verification is a technique that uses mathematical methods to prove the correctness of a design. It is often used to verify that the design satisfies its specification in all possible states.

- **Equivalence Checking: Verifying that the RTL and gate-level designs are functionally equivalent.**
- **Property Checking: Proving that certain properties (e.g., safety or liveness) hold for the design in all cases.**

6.5 Best Practices for Simulation-Based Verification

- **Develop Thorough Testbenches: Use both directed and random testing to ensure a wide range of conditions are covered.**
- **Use Assertions: Integrate assertions into the RTL code to automatically verify design properties.**
- **Ensure High Coverage: Aim for high statement, branch, and functional coverage to ensure comprehensive testing of the design.**
- **Iterate on Simulation Results: Based on simulation findings, refine the design and testbench to catch edge cases and corner scenarios.**
- **Automate Testing: Use tools like CI/CD pipelines to automate verification and ensure continuous testing during the development process.**

6.6 Summary of Key Concepts

- **RTL Verification: The process of verifying the correctness of RTL designs using simulation techniques.**

- **Simulation Types: Functional simulation, timing simulation, and gate-level simulation ensure that designs function correctly under various conditions.**

- **Verification Techniques: Testbenches, assertion-based verification, and coverage help ensure the design meets the specifications and performs correctly.**

- **Methodologies: UVM and formal verification methodologies offer standard approaches to structured, efficient verification.**