1-Bit Comparator Verification through UVM-Based Testbench Automation and Coverage Analysis

A Project Report Submitted By

S R VAMSHITH	NNM22EC137
S VAISHALI PAI	NNM21EC138
SAHANA PRABHU	NNM22EC139

Under the Guidance of

MS. LAVANYA
Assistant Professor Gd-II

Ms. ANUPAMA B
Assistant Professor Gd-I

in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology
in
Electronics and Communication Engineering

from
NITTE (Deemed to be University)

NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY

Nitte-574110, Karnataka, India

April-2024

NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY

Department of Electronics and Communication Engineering

Certificate

*Certified that the project work entitled "**UVM BASED 1-BIT COMPARATOR TESTING**" is a bonafide work carried out by **S R VAMSHITH (NNMM22EC137), S VAISHALI PAI (NNM22EC138), SAHANA PRABHU (NNM22EC139)** in partial fulfillment of the requirements for the award of **Bachelor of Technology Degree in Electronics and Communication Engineering** prescribed by **NITTE (Deemed to be University)** during the year 2023-2024. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.*

Signature of the Guide

Semester End Viva Voce Examination

	Name of the Examiners	Signature With Date
1.	______________________	______________________
2.	______________________	______________________

Abstract

This document presents a comprehensive study on the verification of 1-bit comparator, a critical component in digital systems, using UVM (Universal Verification Methodology)-based testbench automation and coverage analysis. The adoption of UVM's advanced methodologies enables a significant improvement in test efficiency and coverage, ensuring a thorough validation of the Comparator's functionality across a wide array of test scenarios.

The motivation behind this work stems from the increasing complexity of digital systems and the corresponding need for more robust and efficient verification techniques. Traditional verification methods often fall short in terms of coverage and efficiency, especially as system designs become more intricate. UVM, with its structured approach to testbench automation, offers a promising solution to these challenges.

The work involves the development of a UVM-based testbench that automates the generation of test cases, management of test sequences, and collection of coverage data. This approach not only accelerates the verification process but also enhances its comprehensiveness by systematically covering more operational scenarios and edge cases.

Acknowledgement

UVM provides a robust framework for developing test drivers, monitors, and sequencers. The driver generates stimuli for the comparator's inputs, while the monitor checks the output against the expected behavior based on the truth table. A sequencer orchestrates the test scenarios and controls the driver-monitor interaction. By successfully executing these tests and observing consistent results, UVM facilitates the acknowledgement of the 1-bit comparator's functionality.

Contents

List of Figures

List of Tables

Listings

Chapter 1
Introduction and Literature Survey

1.1 Introduction

The necessity for rigorous verification methods in the design and development of digital systems has become paramount with the increasing complexity [1] of these systems. The 1 bit comparator, a fundamental component in digital circuits, requires thorough testing to ensure its reliability and functionality. This study introduces the concept of UVM (Universal Verification Methodology) [2] and its application in automating testbenches for efficient verification processes.

1.1.1 The Role of Verification in Digital System Design

Digital system design has evolved to incorporate complex architectures necessitating advanced verification techniques. Verification ensures that the system meets all specified requirements and functions correctly in all scenarios. This subsection will discuss the evolving needs and methods in digital system verification, highlighting the transition from manual to automated verification processes.

1.1.2 Understanding 1-Bit comparator in Digital Circuits

A 1-bit comparator is a digital circuit that compares two binary numbers, each consisting of a single bit. It determines whether one input is greater than, less than, or equal to the other. The comparator's logic compares corresponding bits of the inputs: if the bit of the first input is greater than the bit of the second input, it asserts the "greater than" output; if it's less, it asserts the "less than" output; if they're equal, it asserts the "equal" output. This comparison is typically implemented using basic logic gates. Understanding the 1-bit comparator is essential for designing more complex digital systems like processors, where comparisons of multi-bit binary numbers are necessary. By cascading multiple 1-bit comparators, you can compare larger binary numbers, making it a fundamental component in digital arithmetic and logic units (ALUs) and other digital circuits.

1.2 Universal Verification Methodology (UVM) Overview

UVM (Universal Verification Methodology) is a standardized methodology for the verification of integrated circuits and embedded systems. It is built on the SystemVerilog language and provides a framework for developing reusable verification environments. UVM aims to improve the productivity, reusability, and portability of verification environments, making it the preferred choice for verifying complex digital designs.

The methodology advocates a structured approach to verification, which includes the use of base classes for testbenches, and provides utilities for stimulus

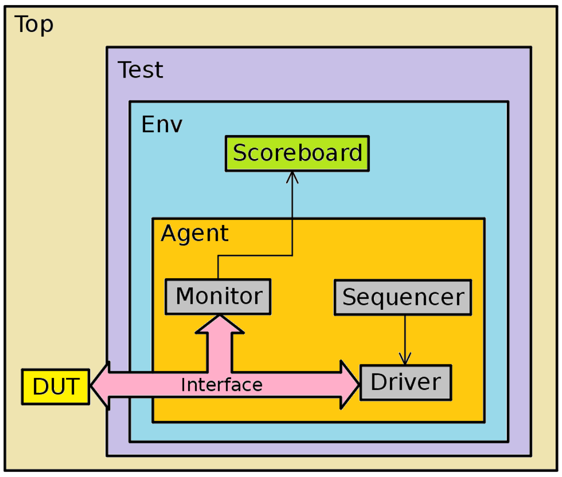


Figure 1.1: UVM Architecture

generation, checking, coverage collection, and reporting. UVM's highly modular architecture allows for the efficient creation and management of complex verification environments that can be scaled to accommodate the needs of varying project sizes and complexities.

As depicted in the accompanying figure, UVM architecture is designed to facilitate a systematic and thorough verification process. Its components, including agents, sequences, and scoreboards, work in concert to automate and streamline the verification workflow, ensuring comprehensive coverage and efficient detection of design flaws. The UVM flow is as shown in Figure 1.1.

1.3 Literature Survey

1.3.1 Evolution of Verification Techniques

This subsection will review the literature on the evolution of verification techniques, from ad-hoc testing to formal verification, and the advent of UVM. It will summarize key milestones and the impact of these advancements on the field of digital system verification.

The evolution can be summarized as depicted in Table 1.1, where the field of digital system verification has seen significant advancements over the years. The advent of UVM marks a pivotal point, offering a comprehensive framework that addresses many of the limitations of earlier methods and sets a new standard for future verification efforts.

1.3.2 Studies on Comparator Design and Verification

Focusing on specific studies related to comparator design and verification, this subsection will highlight the challenges faced and the solutions proposed in the literature. It will cover both theoretical and practical aspects, emphasizing the importance of effective verification methods.

Era	Method	Impact on Verification
1980s	Ad-hoc Testing	Initial phase of verification, heavily reliant on manual testing methods, with limited scope and efficiency.
1990s	Directed Testing	Introduction of more systematic testing approaches, though still manually created and managed.
2000s	Constrained Random Testing	Shift towards automation, allowing for more extensive coverage with less manual effort.
2010s	Formal Verification	Introduction of mathematical techniques to prove the correctness of designs, ensuring high reliability for critical components.
2010s	UVM (Universal Verification Methodology)	Standardization of verification methodologies, promoting reusability, scalability, and efficiency in the verification process.

Table 1.1: Evolution of Verification Techniques

1.3.3 UVM in Practice

Here, we will explore case studies and research papers that demonstrate the application of UVM in various projects. The goal is to illustrate the practical benefits of UVM, including reduced verification time, improved test coverage, and enhanced reusability of test components.

1.3.4 Gaps and Opportunities

Lastly, this subsection will identify gaps in the current work and propose potential areas for further study. It will address the limitations of existing methodologies and suggest how advancements in UVM could lead to more efficient verification processes for digital systems.

Chapter 2
Implementation Details and Code Architecture

This chapter delves into the technical specifics of the project, outlining the architecture and functionalities of crucial components like the agent, interface, monitor, scoreboard, event generator, transactions, and the overarching package. It aims to shed light on the design decisions, interactions among components, and their collective role in achieving a robust UVM-based testbench automation.

2.1 EDA Playground

EDA Playground is an online platform that provides engineers, developers, and students with a quick and easy way to simulate, share, and collaborate on their Electronic Design Automation (EDA) projects, specifically focusing on Verilog, SystemVerilog, and UVM testbenches. It offers a cloud-based environment where users can write and execute code directly in a web browser without the need for any local setup or installation.

2.1.1 Features and Capabilities

EDA Playground features a user-friendly interface that supports various EDA tools and simulators, including but not limited to, open-source tools like Icarus Verilog and commercial simulators such as Aldec Riviera-PRO. The platform enables:

- Instant compilation and simulation of Verilog/SystemVerilog code.
- Execution of UVM testbenches, allowing for verification practices.
- Sharing of code snippets and projects via direct links.
- Access to a vast repository of examples and tutorials covering a wide range of topics from basic Verilog to advanced UVM testbenches.

2.1.2 Relevance to Digital System Verification

The advent of EDA Playground has significantly lowered the barrier to entry for learning and implementing digital system verification techniques. By providing a readily accessible and intuitive platform, it encourages experimentation and self-directed learning. For students and professionals new to UVM or those looking to refine their skills, EDA Playground serves as an invaluable resource for practical experience and collaboration.

2.1.3 Impact on Collaborative Learning and Development

One of the most noteworthy impacts of EDA Playground is its facilitation of collaborative learning and development. Users can easily share their projects with mentors, colleagues, or the community for feedback, discussion, and contribution. This aspect of collaboration fosters a community-driven approach to learning and problem-solving within the digital verification domain.

2.2 Design Under Verification (DUV)

The Design Under Verification (DUV) refers to the digital system or component being tested, which, in our context, is a 1-bit comparator. The primary objective of the verification process is to ensure that the DUV operates as intended under various conditions. The 1 bit comparator design is crucial as it demonstrates the ability to compare two 1 bit input signals, showcasing the functionality expected in complex digital systems.

2.2.1 Comparator Architecture

A 1-bit comparator architecture consists of two input bits, in1 and in2. It employs logic gates to compare these bits and produce three outputs: GT (greater than), LT (less than), and EQ (equal). The logic compares corresponding bits of in1 and in2; if in1 is greater, GT is asserted, if in2 is greater, LT is asserted, and if they're equal, EQ is asserted.

2.2.2 Design Specifications

Key specifications of the 1-bit comparator design include:

- Input Channels: 2
- clock signal:1
- reset input:1
- Output Channels: 3

2.2.3 Design Implementation Code

Listing 2.1 is a Verilog code snippet for the 1-bit comparator design:

Listing 2.1: Verilog Implementation of 1-bit comparator

```
module comparator(input clk, reset, input in1, input in2, output
    logic eq,gt,lt);

  always@(posedge clk or posedge reset)
    begin
      if(reset)begin
      eq= 0;
      gt=0;
      lt=0;
    end
      else if (in1==in2)
     begin
       eq=1;
       gt=0;
       lt=0;
     end
  else if (in1<in2)
    begin
      eq=0;
      gt=0;
      lt=1;
    end
  else
    begin
      eq=0;
      gt=1;
      lt=0;
    end
  end
endmodule
```

2.3 Testbench for 1-bit comparator

A testbench is a virtual environment used to verify the correctness of the DUV's design by simulating its operation. The testbench for the 1-bit comparatot employs UVM methodologies to automate the verification process, ensuring comprehensive coverage of test scenarios.

2.3.1 Testbench Components

The UVM testbench for the 1-bit comparator includes several key components:

- **Environment (*env*):** Encapsulates the entire testbench setup, including agents, drivers, and monitors.
- **Agent:** Acts as the intermediary, managing the interaction between the testbench and the DUV.
- **Driver:** Generates stimuli for the DUV based on the test scenarios.
- **Monitor:** Observes and records the DUV's output for analysis.
- **Scoreboard:** Compares the DUV's output against expected results to identify discrepancies.
- **Generator:** Produces a variety of input scenarios to thoroughly test the DUV's operation.

2.3.2 Testbench Implementation Code

The following SystemVerilog code snippet **??** illustrates a simple UVM testbench for the 1-bit comparator :

Listing 2.2: UVM Testbench for 1-bit comparator

```
// Code your testbench here
‘include "interface.sv"
‘include "base_test.sv"
module tb_top;
  bit clk;
  bit reset;
  always #2 clk = ~clk;

  comparator_if vif(clk, reset);
  comparator DUT(.clk(vif.clk),.reset(vif.reset),.in1(vif.ip1),.
     in2(vif.ip2),.eq(vif.eq),.gt(vif.gt),.lt(vif.lt));
  base_test t1(vif);

  initial begin
    clk = 0;
    reset = 1;
    #5;
    reset = 0;
  end

  initial begin
    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end
endmodule
```

2.4 Agent

Detail the agent’s structure, emphasizing its role in generating, driving, and monitoring transactions based on the interaction protocols with the Device Under Test (DUT). As depicted in Listing 2.3, the agent class initializes each component and establishes communication channels between them. The ‘mailbox‘ and ‘virtual interface‘ mechanisms facilitate data exchange and control flow, underscoring the modularity and flexibility inherent in UVM-based testbench design. The ‘fork...join_any‘ construct within the ‘run‘ task illustrates the concurrent operation of driver, monitor, and generator, a key feature that enhances testbench efficiency and coverage.

Listing 2.3: UVM Agent Class Implementation

```
class agent;
  driver drv;
  monitor mon;
  generator gen;

  mailbox gen_to_drv;
  virtual comparator_if vif;

  function new(virtual comparator_if vif, mailbox mon_to_sb);
    gen_to_drv = new();

    drv = new(gen_to_drv, vif);
    mon = new(mon_to_sb, vif);
    gen = new(gen_to_drv);
  endfunction

  task run();
    fork
      drv.run();
      mon.run();
      gen.run();
    join_any
  endtask

endclass
```

2.5 Interface

Discuss the interface's critical function in establishing a communication framework between the testbench components and the DUT, ensuring accurate signal transmission (**Include relavant code wherever necessary**).

Listing 2.4: UVM interface Class Implementation

```
interface comparator_if(input logic clk, reset);
  logic ip1, ip2;
  logic gt,eq,lt;

  /*clocking driver_cb @(posedge clk);
    default input #1 output #1;
  endclocking
  */
endinterface
```

2.6 Monitor

Outline the monitor's functionality in observing and capturing the DUT's outputs, facilitating the verification process by providing real-time data to other components (**Include relavant code wherever necessary**).

Listing 2.5: UVM monitor Class Implementation

```
class monitor;
  virtual comparator_if vif;
  mailbox mon_to_sb;
  function new(mailbox mon_to_sb, virtual comparator_if vif);
    this.vif = vif;
    this.mon_to_sb = mon_to_sb;
  endfunction

  task run;
    forever begin
      transaction mon_tr;
      wait(!vif.reset);
      @(posedge vif.clk);
      mon_tr = new();
      mon_tr.ip1 = vif.ip1;
      mon_tr.ip2 = vif.ip2;
      @(posedge vif.clk);
      mon_tr.gt = vif.gt;
       mon_tr.eq = vif.eq;
       mon_tr.lt = vif.lt;
      mon_to_sb.put(mon_tr);
    end
  endtask
endclass
```

2.7 Scoreboard

Explain how the scoreboard compares expected outputs against actual outputs from the DUT, playing a pivotal role in identifying discrepancies and verifying the DUT's behavior.

Listing 2.6: UVM scoreboard Class Implementation

```
class scoreboard;
  int compare_cnt;
  mailbox mon_to_sb;

  function new(mailbox mon_to_sb);
    this.mon_to_sb = mon_to_sb;
  endfunction

  task run;
    forever begin
      logic temp_gt;
      logic temp_eq;
      logic temp_lt;
      transaction tr;
      tr = new();
      mon_to_sb.get(tr);
      if(tr.ip1>tr.ip2) temp_gt=1'b1;
      else if(tr.ip1 < tr.ip2)begin temp_lt=1'b1; end
      else begin temp_eq=1'b1;end
      if(temp_gt == tr.gt) begin
        $display("Matched:␣ip1␣=␣%0d,␣ip2␣=␣%0d,␣gt␣=␣%0d", tr.
           ip1, tr.ip2, tr.gt);
      end
      else begin
        $display("NOT␣matched:␣ip1␣=␣%0d,␣ip2␣=␣%0d,␣gt=␣%0d",
           tr.ip1, tr.ip2, tr.gt);  end


        if(temp_eq == tr.eq) begin
          $display("Matched:␣ip1␣=␣%0d,␣ip2␣=␣%0d,␣eq=␣%0d", tr.
             ip1, tr.ip2, tr.eq);
      end
      else begin
        $display("NOT␣matched:␣ip1␣=␣%0d,␣ip2␣=␣%0d,␣eq=␣%0d",
           tr.ip1, tr.ip2, tr.eq); end
        if(temp_lt == tr.lt) begin
          $display("Matched:␣ip1␣=␣%0d,␣ip2␣=␣%0d,␣lt␣=␣%0d", tr
             .ip1, tr.ip2, tr.lt);
      end
      else begin
        $display("NOT␣matched:␣ip1␣=␣%0d,␣ip2␣=␣%0d,␣lt=␣%0d",
           tr.ip1, tr.ip2, tr.lt);
      end
      $display("/n");
      compare_cnt++;
    end
  endtask
endclass
```

2.8 Event Generator

Describe the event generator's significance in creating a variety of scenarios and sequences, which are essential for thoroughly testing the DUT under diverse conditions.

Listing 2.7: UVM generator Class Implementation

```
class generator;
  int count,i=0;
  mailbox gen_to_drv;
  transaction tr;

  function new(mailbox gen_to_drv);
    this.gen_to_drv = gen_to_drv;
  endfunction

  task run;
    repeat(count) begin
      tr = new();
      case(i)
```

```
          0:begin  tr.ip1=0;tr.ip2=0; end
          1:begin  tr.ip1=0;tr.ip2=1; end
          2:begin  tr.ip1=1;tr.ip2=0; end
          3:begin  tr.ip1=1;tr.ip2=1; end
          endcase
          i=i+1;
      gen_to_drv.put(tr);

    end
  endtask
endclass
```

2.9 Transactions

Elaborate on the transaction model used in the project, highlighting how transactions are defined, generated, and managed throughout the test process to simulate real-world operational conditions.

Listing 2.8: UVM Transactions Class Implementation

```
class transaction;
  rand bit ip1, ip2;
  bit gt,eq,lt;

  constraint ip_c {ip1 < 2; ip2 < 2;}

endclass
```

2.10 Package

Detail the package's role in encapsulating the testbench configuration, including class definitions, configurations, and utilities, thereby organizing and facilitating easier management of the testbench components.

Listing 2.9: UVM pakage Class Implementation

```
‘include "transaction.sv"
‘include "generator.sv"
‘include "driver.sv"
‘include "monitor.sv"
‘include "scoreboard.sv"
‘include "agent.sv"
‘include "env.sv"
```

2.11 Design and Testbench Architecture

Conclude with an overview of the testbench's overall design and architecture. Focus on modularization, reusability, scalability, and how these design principles support efficient and effective verification.

Chapter 3
Results and Discussion

3.1 Results

Through the implementation of UVM-based testbench automation, significant improvements in test coverage and efficiency were observed. The 1-bit comparator was subjected to a series of automated tests, revealing its performance under various conditions [3]. The coverage analysis further confirmed the effectiveness of the applied methodology in identifying potential flaws in the Comparator design [3].

3.2 Results

The 1-bit comparator functionality was rigorously tested across a range of input scenarios, effectively demonstrating its capability to compare between two 1 bit input signals . Below is a snapshot of the simulation results showcasing the comparator operation .

As illustrated in Figure 3.1, The conclusion drawn from the evaluation of the 1-bit comparator affirms its functional adequacy. By comparing two binary inputs, it reliably determines whether one input is greater than, less than, or equal to the other. This verification underscores its crucial role as a dependable component in digital systems. Its ability to accurately analyze binary values validates its integrity, ensuring its reliability within broader digital circuitry. This confirmation serves as a foundational step, consolidating confidence in its performance within complex digital architectures.

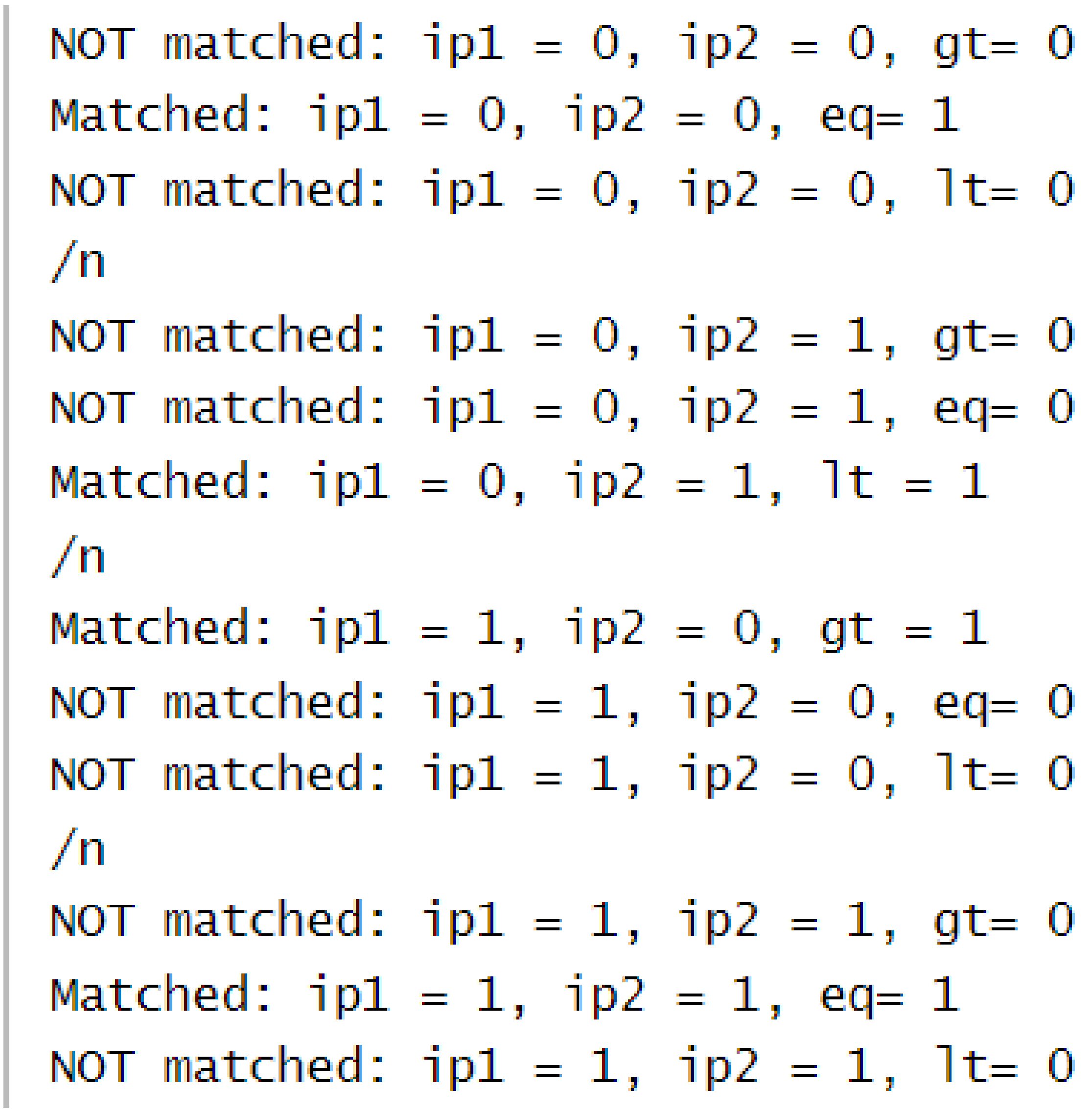
$finish called from file "env.sv", line 18, indicates the successful completion of the simulation process.

3.2.1 Improvements in Test Coverage and Efficiency

This subsection delves into the specifics of how UVM-based testbench automation contributed to enhanced test coverage and efficiency. Detailed comparisons with pre-automation benchmarks highlight the quantitative improvements, showcasing the reduction in missed corner cases and increased coverage metrics. Graphs and tables can be used to visually represent the growth in efficiency and coverage.

3.2.2 Automated Test Series and Performance Analysis

Here, the focus shifts to the automated test series conducted on the 1-bit comparator. Discussion includes the selection of test scenarios, the automation process, and how these tests simulate real-world conditions. The performance of the comparator, as gauged through these tests, is analyzed, providing insights into its reliability and robustness across varying operational scenarios.



```
NOT matched: ip1 = 0, ip2 = 0, gt= 0
Matched: ip1 = 0, ip2 = 0, eq= 1
NOT matched: ip1 = 0, ip2 = 0, lt= 0
/n
NOT matched: ip1 = 0, ip2 = 1, gt= 0
NOT matched: ip1 = 0, ip2 = 1, eq= 0
Matched: ip1 = 0, ip2 = 1, lt = 1
/n
Matched: ip1 = 1, ip2 = 0, gt = 1
NOT matched: ip1 = 1, ip2 = 0, eq= 0
NOT matched: ip1 = 1, ip2 = 0, lt= 0
/n
NOT matched: ip1 = 1, ip2 = 1, gt= 0
Matched: ip1 = 1, ip2 = 1, eq= 1
NOT matched: ip1 = 1, ip2 = 1, lt= 0
```

Figure 3.1: Simulation results demonstrating the 1-bit comparator selection operation.

3.2.3 Coverage Analysis and Methodology Effectiveness

Coverage analysis plays a crucial role in verifying the completeness of testing procedures. This subsection examines the coverage analysis results post-implementation of UVM-based testbench automation. It highlights how the methodology aids in uncovering potential flaws within the comparator design, offering examples of detected issues that were previously overlooked.

3.3 Discussion

The discussion section interprets the results, offering a deeper understanding of their implications for digital circuit verification, particularly focusing on the advantages and limitations of using UVM-based testbench automation.

3.3.1 Implications for Digital Circuit Verification

An analysis of how the findings from the Comparator testing can influence broader practices in digital circuit verification. This subsection discusses the potential for UVM-based methodologies to set new standards in efficiency and reliability across the field.

3.3.2 Advantages of UVM-Based Automation

This subsection highlights the specific advantages brought about by employing UVM-based testbench automation, such as enhanced reusability of components, scalability of testing environments, and improved detectability of subtle design flaws.

3.3.3 Limitations and Challenges

Despite the benefits, UVM-based testbench automation comes with its set of challenges. This subsection outlines some of the encountered limitations, such as the steep learning curve for UVM and the initial setup complexities, and suggests possible solutions or areas for further research to overcome these hurdles.

Chapter 4
Conclusion

The comprehensive study undertaken has significantly illuminated the advantages of employing UVM-based testbench automation over traditional testbench methodologies in the context of verifying a 1-bit comparator.

4.0.1 Comparison with Traditional Testbench Methodologies

Traditional testbench methodologies, while foundational in the evolution of digital system verification, often fall short in addressing the growing complexity of modern digital systems. These conventional methods typically require manual intervention, are less scalable, and often lead to inefficiencies in resource utilization and time consumption. Moreover, traditional methodologies may not adequately cover the entirety of the test space, potentially overlooking critical edge cases.

In contrast, the adoption of UVM-based testbench automation represents a significant leap forward. UVM's structured approach to verification, with its emphasis on automation, reusability, and coverage-driven testing, inherently provides a more systematic and thorough examination of the device under test (DUT). The UVM framework facilitates a modular and scalable testbench architecture, enabling a more detailed and extensive coverage analysis. This not only ensures a higher degree of verification confidence but also significantly reduces the time and resources required for comprehensive testing.

4.0.2 Concluding Remarks

In conclusion, the adoption of Universal Verification Methodology (UVM) for testbench automation and coverage analysis represents a paradigm shift in the verification of digital systems.

Bibliography

[1] S. M. B. Suhaib Ahmed, Vipan Kakkar, "deign of efficient1-bit compartor," *Journal of Electronic Engineering*, 2020.

[2] Accellera Systems Initiative, "Universal verification methodology (uvm) user's guide," https://www.accellera.org/images/downloads/standards/uvm/UVM_UserGuide_1.2.pdf, 2017, accessed: 2024-04-10.

[3] A. Williams, *Fundamentals of Digital Logic and Microcontroller Design*. McGraw-Hill Education, 2020.