ALU VERIFICATION DOCUMENT

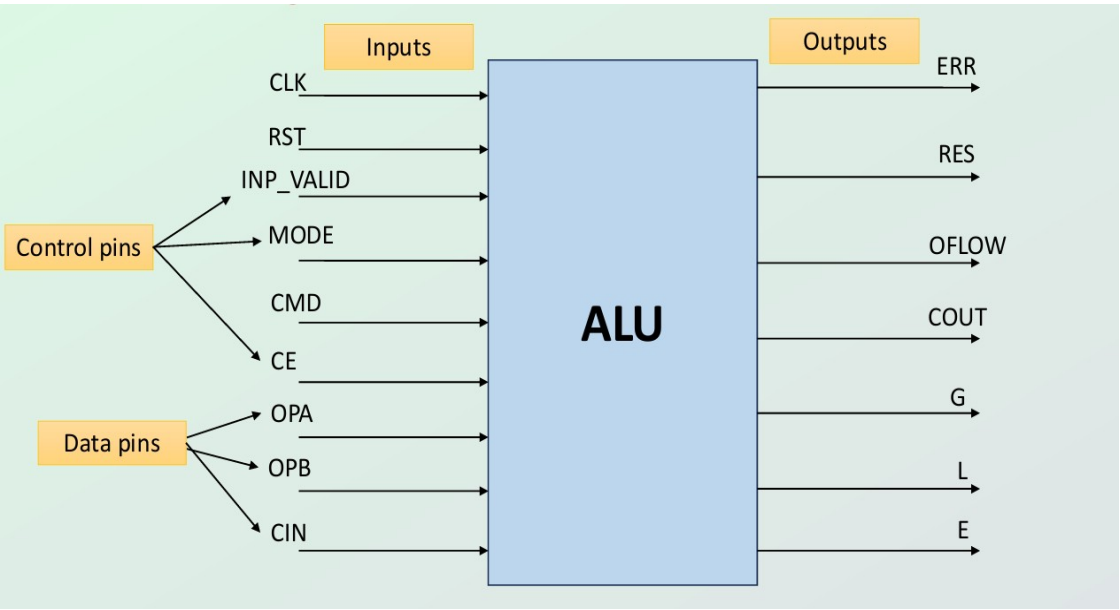
Tejas Poojary
Employee ID:6119
Mirafra Technologies
Manipal

Table of contents

Project Overview

Arithmetic and Logic Unit(ALU) :

The Arithmetic Logic Unit (ALU) is a core component of any processing system, performing both arithmetic and logical operations on input data. This project focuses on verifying a flexible and parameterized ALU design in System Verilog, capable of handling a wide variety of instructions including unsigned arithmetic, logical operations, shift and rotate instructions.



The ALU verification testbench thoroughly checks the ALU design by testing all different arithmetic and logical commands. For each command,it verifies that the correct result is produced and all output flags work properly, including overflow, carry-out, comparison flags, and error signals. Additionally,it verifies

timing requirements by ensuring multiplication operations complete in exactly 3 clock cycles and other operations complete in 2 clock cycles.It also checks whether the 16-cycle timeout works correctly. Finally, the testbench runs with different data sizes (16-bit, 32-bit, 64-bit, 128-bit) to ensure the parameterized design works correctly at all supported widths.

Verification Objectives

Basic Functionality Verification

Arithmetic Operations (Mode = 1)

- Verify all arithmetic commands (0-10): ADD, SUB, ADD_CIN, SUB_CIN, INC_A, DEC_A, INC_B, DEC_B, CMP, multiply variants.
- Test carry-in/carry-out functionality.
- Verify overflow detection (OFLOW).
- Test comparison flags (G, L, E).

Logical Operations (Mode = 0)

- Verify all logical commands (0-13): AND, NAND, OR, NOR, XOR, XNOR, NOT operations.
- Test shift operations (SHR1, SHL1).
- Verify rotation operations (ROL_A_B, ROR_A_B).
- Test error generation for invalid rotation parameters.

Control Signal Verification

- Verify CLK, RST (asynchronous reset) functionality.
- Test CE (clock enable) operation.
- Verify INP_VALID signal combinations (00, 01, 10, 11).
- Test MODE signal switching between arithmetic and logical operations.

Error Handling and Edge Cases

- Verify OFLOW (overflow) detection for arithmetic operations.
- Test ERR signal generation for invalid rotation parameters.
- Verify 16 clock cycle timeout behavior.
- Test boundary conditions and wrap-around cases.

Parameterization Testing

- Test different data widths (16, 32, 64, 128 bits).
- Verify result width scaling (parameterized + 1 bits).
- Test command width parameterization.

Timing Verification

- Verify 3-cycle operation processing time for multiplication.
- Verify 2-cycle operation time for other operations.
- Verify output timing and signal dependencies.

Coverage Goals

- Achieve 100% functional coverage for all commands and modes.
- Complete code coverage (line, branch, toggle).
- Cover all error conditions and edge cases.
- Test all operand valid combinations.

DUT Interfaces

The Design Under Test (DUT) in SystemVerilog verification is the hardware design or module being verified for correctness and functionality. It represents the actual RTL (Register Transfer Level) code that implements the desired digital circuit or system, written in hardware description languages like Verilog or SystemVerilog. The DUT sits at the center of the verification environment, receiving stimulus from testbenches, drivers, and verification components while producing responses that are monitored and checked by scoreboards and checkers.

The goal is to find any bugs or problems in the DUT before the hardware is actually built, since fixing errors after manufacturing is extremely expensive.

Sl.No	Pin Name	Direction	No. Of bits	Function
1	OPA	Input		Parameterized operand 1
2	OPB	Input		Parameterized operand 2
3	CIN	Input	1	This is the active high carry in input signal of 1-bit
4	CLK	Input	1	This is the clock signal to the design and it is edge sensitive

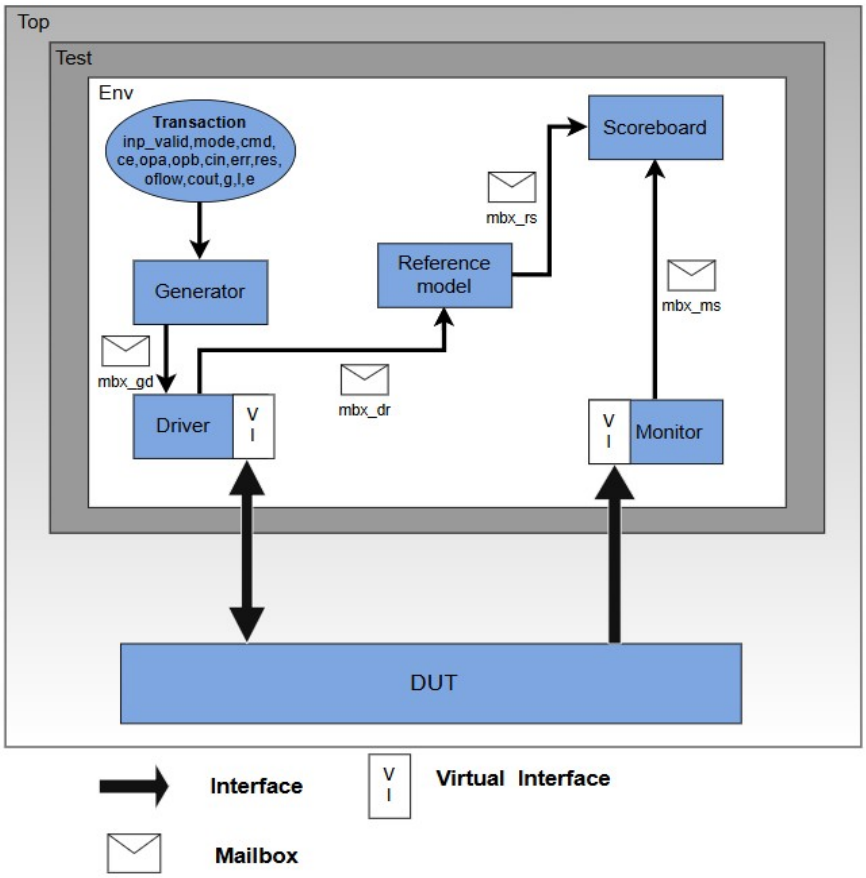
5	RST	Input	1	This is the active high asynchronous reset to the design
6	CE	Input	1	This is the active high clock enable signal 1 bit
7	MODE	Input	1	MODE signal 1 bit is high, then this is an Arithmetic Operation otherwise it is logical operation
8	INP_VALID	Input	2	Operands are valid as per below table 00: No operand is valid 01: Operand A is valid 10: Operand B is valid 11: Operand A and B both
9	CMD	Input		Parameterized (4 bit default) Arithemetic Commands CMD = 0 :ADD , 1: SUB, 2: ADD_CIN, 3: SUB_CIN, 4 :INC_A , 5:DEC_A , 6: INC_B 7: DEC_B, 8: CMP, 9: Operand A and B both incr by 1 and then multiplication performed. 10: Operand A left shift by 1 and then multiply

				with B. Logical Commands CMD = 0: AND, 1: NAND, 2: OR, 3: NOR, 4: XOR, 5: XNOR, 6: NOT_A, 7: NOT_B, 8: SHR1_A, 9: SHL1_A, 10: SHR1_B, 11: SHL1_B, 12:ROL_A_B if operandB [7:4] any bit is 1 then its error whereas output will be as per [2:0 as mentioned above]. 13:ROR_A_B. if operandB [7:4] any bit is 1 then its error whereas output will be as per [2:0 as mentioned above].
10	RES	Output		This is the total parameterized plus 1 bits result of the instruction performed by the ALU.

11	OFLOW	Output	1	This 1-bit signal indicates an output overflow, during Addition/Subtraction
12	COUT	Output	1	This is the carry out signal of 1-bit, during Addition/Subtraction
13	G	Output	1	This is the comparator output of 1-bit,which indicates that the value of OPA is greater than the value of OPB
14	L	Output	1	This is the comparator output of 1-bit,which indicates that the value of OPA is lesser than the value of OPB
15	E	Output	1	This is the comparator output of 1-bit,which indicates that the value of OPA is equal to the value of OPB
16	ERR	Output	1	When Cmd is selected as 12 or 13 and mode is logical operation , if 4th ,5th ,6th and 7th bit of OPB are 1, then ERR bit will be 1 else it is high impedance.

Testbench Architecture

Testbench Architecture is a structured verification framework that uses classes, mailboxes, and virtual interfaces to create a reusable verification environment that can thoroughly test a design's functionality through automated stimulus generation and result checking.



A typical testbench includes several key components: a stimulus generator (usually in the form of drivers), monitors to observe DUT outputs, scoreboards for result checking, and assertions for protocol validation

SystemVerilog Testbench Architecture Components

Transaction Class

It holds all input and output signals of the Design Under Test (DUT), except clock and reset signals which are created in the top module. The transaction class can have constraints to limit the input values.It is built using a class structure.

Generator Component

It makes random test data (packets) for the DUT. The generator sends the created data to the driver through a mailbox. It is written as a class.

Driver Component

It drives the generated signals into actual pin signals at DUT inputs. It gets packets from the generator through a mailbox and sends them to the DUT using a virtual interface based on the DUT specifications.It also sends the packets to the reference model through another mailbox. It is also written as a class.

Monitor Component

It captures the pin signals from the DUT outputs into packets. Collects DUT output signals through a virtual interface and sends them to the scoreboard using a mailbox.It is also written as a class.

Reference Model

A 'golden reference' model that works like the DUT and creates expected results for checking against actual results.This model helps us study the system operations and mimic athe actual behaviour of the design under verification. It takes inputs from driver through a mailbox and sends outputs to scoreboard through another mailbox.

Scoreboard Component

It gets packets (expected results) from the reference model through a mailbox.On the other side it gets packets (actual results) from the monitor through another mailbox. Compares expected and actual results and creates a comparison report and gives pass or fail result after comparison.

Environment Component

It creates instances of generator, driver, monitor, reference model and scoreboard. It builds and connects all the parts together properly. The instances of mailboxes are also created inside environment and passed to different component instances.

Test Component

This is a component where different test cases are written and executed. The environment is created and set up here.The test component gets the interface from the top module.

Top Module

The highest module in the testbench where control signals like clock and reset are created and passed. It is written as a module where the interface, DUT and test instances are created. A specific test is run from inside an initial block.

Mailbox

Mailboxes in System Verilog are communication channels that enable safe data transfer between different testbench components running as separate threads. They function as FIFO (First-In-First-Out) queues where one component can deposit data using the put() method while another component retrieves it using the get() method. Mailboxes support both blocking operations and non-blocking operations.

Virtual Interface

Virtual interfaces in SystemVerilog provide a mechanism for testbench classes to access and manipulate design signals without direct module instantiation.Since classes cannot directly instantiate modules or interfaces, virtual interfaces serve as a bridge between the object-oriented testbench environment and the module-based DUT.Driver uses virtual interfaces to change transaction data into actual pin signals and sends them to the DUT whereas Monitor uses it to read pin signals from the DUT and change them into transaction data.

Test Plan

A Test Plan is a comprehensive document that outlines the complete verification strategy for a design. It serves as a roadmap that defines what needs to be tested, how it will be tested, and what criteria determine verification success.

A well-structured test plan includes a verification plan,assertion plan and a coverage plan.

Verification Plan

The Verification Plan forms the core section of the test plan and defines the functional verification strategy. It includes detailed descriptions of all features and functionalities that need to be verified, along with the verification approach for each.

Link:https://in.docworkspace.com/d/sIPz5tOpM68P5wwY?sa=601.1037

Assertion Plan

The Assertion Plan defines the strategy for using assertions to verify design properties and protocols. It specifies what design behaviors, interfaces, and internal conditions will be monitored using System Verilog Assertions (SVA).This plan identifies critical design properties that must always hold true during operation.

Link:https://in.docworkspace.com/d/sIEX5tOpMnsf5wwY?sa=601.1037

Coverage plan

Functional coverage in SystemVerilog verification is a metric-driven approach to measure how thoroughly the design's functionality has been exercised during simulation. It uses covergroups, coverpoints, and cross-coverage to define and track specific scenarios, corner cases,that must be tested.

Unlike code coverage which measures structural elements, functional coverage focuses on design intent and specification requirements.

Link:https://in.docworkspace.com/d/sIF35tOpM0ZD6wwY?sa=601.1037