ALU Verification Plan

**VERIFICATION DOCUMENT- ALU**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **CONTENTS** | **PAGE NUMBER** |
| **1** | **Chapter -1** | **3 -7** |
| **2** | **Chapter-2** | **7-15** |

**CHAPTER 1 – DESIGN OVERVIEW**

1 **ALU:-**

Arithmetic Logic Unit (ALU) is a simple combinational circuit, where simple arithmetic, as well as logic operations, are performed in a digital system. During design, the ALU is created with the assistance of a hardware description language, Verilog, typically used for modeling a digital system.

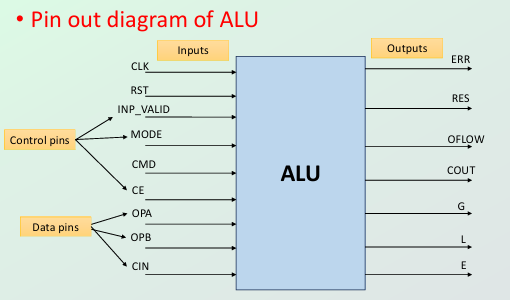
**ALU significance:-**  
• Everything your computer calculates goes through the ALU  
• Computers cannot perform math nor make decisions without ALU  
• Executes incredibly fast - millions of operations every second

* 1. Pros of ALU:  
     • Can do basic math like adding, subtracting, multiplying, and dividing  
     • Helps make logical choices like AND, OR, NOT  
     • Can compare numbers to check which is bigger, smaller, or equal  
     • It's fast, correct, and takes up little space in the processor

## Cons of ALU:

1. **No Memory**  
   • Can’t remember past calculations
2. **Speed Limit**  
   • Can’t go faster than the processor itself
3. **High Power Use**  
   • Uses more power when solving complex problems
   1. Where ALU Is Used:  
      • **Microprocessors & Microcontrollers:** Does math and logic tasks  
      • **Digital Signal Processors (DSPs):** Used for tasks like filtering and sound/image changes  
      • **Embedded Systems:** Helps machines do control and automation work  
      • **Cryptography:** Used in data encryption with special math  
      • **AI & SIMD Units:** Helps with fast math in AI and graphics by working on many values at once

**Project Overview of ALU:**-

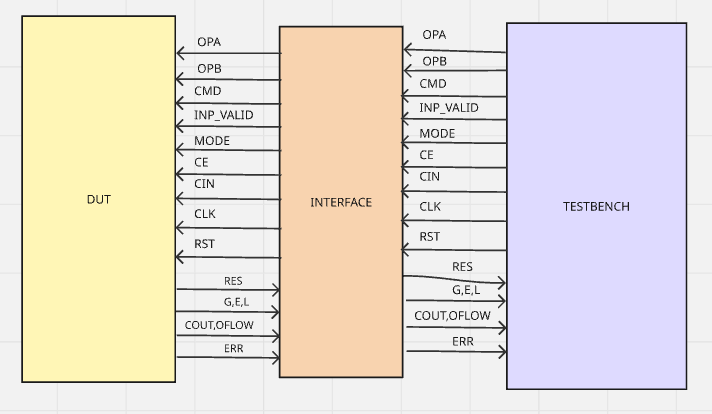


1. This project builds a flexible ALU using Verilog that can do many operations like math, logic, comparing values, and shifting/rotating bits. It is tested with SystemVerilog using a testbench that checks its behavior using coverage and rules (assertions).
2. The pin-out diagram shows how the ALU connects to other parts. It has separate input and output pins to help data move smoothly and control the operation. Control pins decide what the ALU should do and when, while data pins give the numbers to work with. The output shows the result and extra info like comparison results or errors.
3. This kind of design makes the ALU easy to reuse, modify, and connect to bigger systems like CPUs and digital signal processors.

1.5  **Design Features:-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Serial no** | **Pin name** | **Direction** | **No of bits** | **Function** |
| 1 | OPA | INPUT | Parametrized | Parameterized operand 1 |
| 2 | OPB | INPUT | Parametrized | 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 a Logical Operation |
| 8 | INP\_VALID | INPUT | 2 | Operands are valid as per below table: 00: No operand is valid01: Operand A is valid10: Operand B is valid |
| 9 | RES | OUT | Parameterized +1 | This is the total parameterized plus 1 bits result of the instruction performed by the ALU |
| 10 | OFLOW | OUT | 1 | This 1-bit signal indicates an output overflow during Addition/Subtraction |
| 11 | COUT | OUT | 1 | This is the carry out signal of 1-bit during Addition/Subtraction |
| 12 | G | OUT | 1 | This is the comparator output of 1-bit which indicates that the value of OPA is greater than the value of OPB |
| 13 | L | OUT | 1 | This is the comparator output of 1-bit which indicates that the value of OPA is less than the value of OPB |
| 14 | E | OUT | 1 | This is the comparator output of 1-bit which indicates that the value of OPA is equal to the value of OPB |
| 15 | ERR | OUT | 1 | When CMD is selected as 12 or 13 and mode is logical operation, if 4th, 5th, 6th, and 7th bits of OPB are 1, then ERR bit will be 1; else it is high impedance |

1.6 **Design diagram with interface signals:-**



|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Direction (in clocking block)** | **Bit Width / Size** | **Description** |
| OPA | output | WIDTH bits | Operand A |
| OPB | output | WIDTH bits | Operand B |
| CMD | output | CMD\_WIDTH bits | Operation command |
| IN\_VALID | output | 2 bits | Indicates which operands are valid |
| MODE | output | 1 bit | Arithmetic (1) or logical (0) mode |
| CE | output | 1 bit | Clock enable |
| CIN | output | 1 bit | Carry-in input |

-**Driver**

-**Monitor Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Direction (in clocking block)** | **Bit Width / Size** | **Description** |
| RES | input | WIDTH + 1 bits | Result of ALU operation |
| ERR | input | 1 bit | Error flag for illegal rotate etc. |
| OFLOW | input | 1 bit | Overflow flag |
| COUT | input | 1 bit | Carry-out flag |
| G | input | 1 bit | Comparator: OPA > OPB |
| E | input | 1 bit | Comparator: OPA == OPB |
| L | input | 1 bit | Comparator: OPA < OPB |

-**DUT Interface**

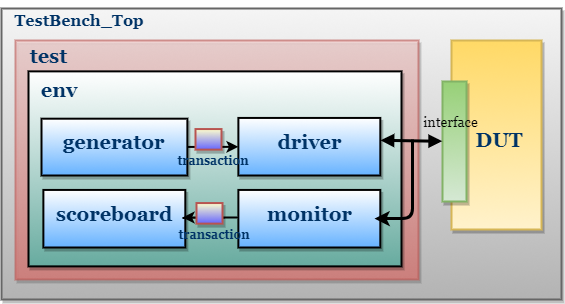
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Direction (in clocking block)** | **Bit Width / Size** | **Description** |
| OPA | input | WIDTH bits | Operand A |
| OPB | input | WIDTH bits | Operand B |
| CMD | input | CMD\_WIDTH bits | Operation command |
| IN\_VALID | input | 2 bits | Indicates which operands are valid |
| MODE | input | 1 bit | Arithmetic (1) or logical (0) mode |
| CE | input | 1 bit | Clock enable |
| CIN | input | 1 bit | Carry-in input |

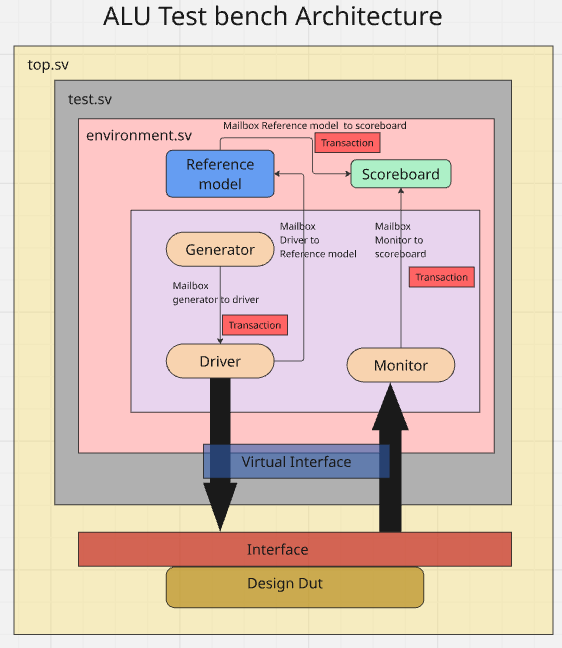
|  |  |  |  |
| --- | --- | --- | --- |
| RES | output | WIDTH + 1 bits | Result of ALU operation |
| ERR | output | 1 bit | Error flag for illegal rotate etc. |
| OFLOW | output | 1 bit | Overflow flag |
| COUT | output | 1 bit | Carry-out flag |
| G | output | 1 bit | Comparator: OPA > OPB |
| E | output | 1 bit | Comparator: OPA == OPB |
| L | output | 1 bit | Comparator: OPA < OPB |

CHAPTER 2 - Verification Architecture

2.1 **Verification Architecture :-**

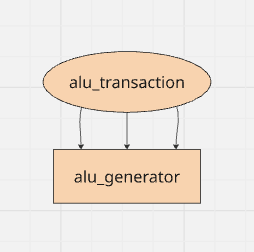
**General Testbench Architecture:**



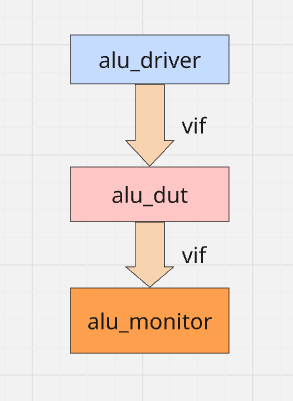
**Proposed Testbench Architecture:**

**Flow chart :**

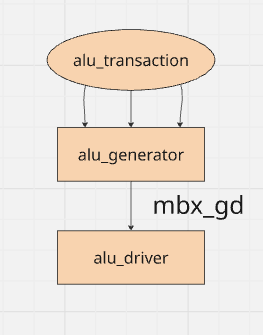
1. **Transaction:**

****

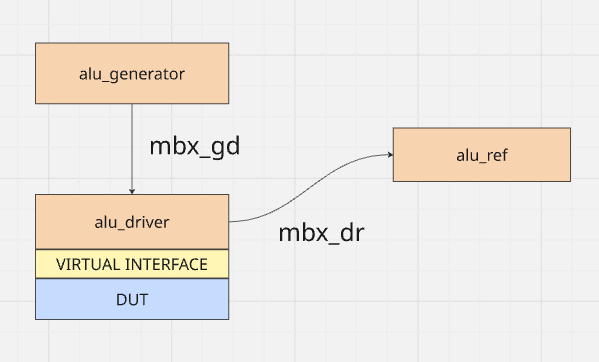
1. In the transaction we have declared all the fields of the design both inputs and outputs and size of the signals .
2. We also declared functions like copy that we use the deep copy to copy all the signals from the class transaction and sent that copied transaction to where we want using mailbox.
3. We use this class to generate random stimulus in the generator.
4. **Interface:**
5. We use the interface to share common bunch of signals. And use them in between different components for connection.
6. In this we declare interface between the driver, monitor and dut.
7. We also wrote the mod ports to specify the direction of the signals and clocking block for synchronization between components.



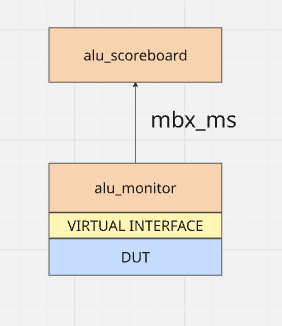
1. **Generator:**
2. In generator we randomize all the stimulus and send to driver.
3. According to design we want to keep the mode and ce and cmd same when we got two operand operation cmd and if the input valid is not 2’b11 then we have to wait for 16 clock cycles if it become 2’b11 before 16 cycles we will turn on the rand mode until we will turn off the rand mode of the ce,mode,cmd.
4. The generated stimulus is sent to driver through the mbx\_gd mailbox of type transaction class.
5. I also used semaphore to synchronize the driver and the generator.

****

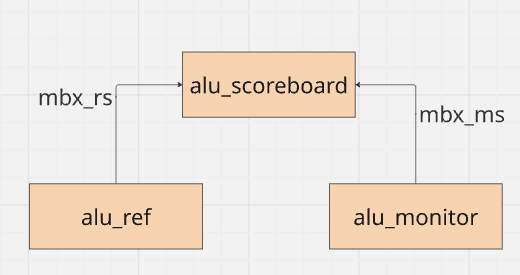
1. **Driver:**

****

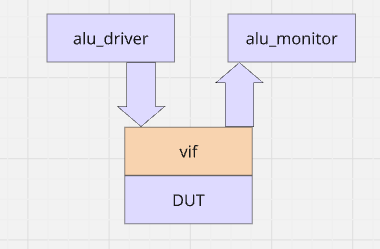
1. There are two mail boxes in the driver one is to get the stimulus from the generator and another one to send the stimulus from driver to reference model using the mail box mbx\_dr.
2. We use the virtual interface to interact with DUT and send the signals to the design using interface only.
3. **Monitor:**

****

1. In the monitor we have Interface where we will read the outputs of the design and send that to scoreboard using the mailbox mbx\_ms.
2. There is much logic in the monitor but to synchronize with the test driver make it difficult.
3. **Scoreboard:**

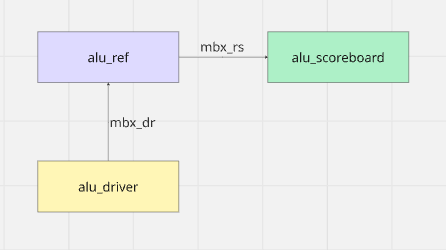
****

1. In the score board we have 2 mailboxes(mbx\_ms,mbx\_rs).
2. one mail box is used to get the outputs from the monitor and another one is used to get the output from the reference model .
3. The main functionality of the scoreboard is to compare the values of the dut driven that we get from the monitor and the reference model give result according to that we evaluate our design.
4. **DUT:**

****

The dut is the design instance created in the top module and it interacts with test bench through the virtual interface (vif) and the outputs are captured in the monitor and the inputs are sent to the dut using the driver.

1. **Reference Model:**

****

1. The reference model has 2 mailboxes (mbx\_rs ,mbx\_dr).
2. They are used to mbx\_dr to drive the stimulus from the driver to the reference model and mbx\_rs is used to send the result from the reference model to the scoreboard.
3. The reference model is the golden model of the design.