

PROJECT: System Verilog Verification of Arithmetic Logic Unit

- **ALU design properties:**

1. Inputs:

This design unit is responsible for executing arithmetic, logic, and shift operation on two 8-bit unsigned input signals, `a_in` and `b_in`. The operation control input, `op_in`, is 3 bits wide and selects one of eight operations as follows:

```
3'b000: result_out = a_in + b_in    // Addition
3'b001: result_out = a_in - b_in    // Subtraction
3'b010: result_out = a_in & b_in    // Bitwise AND
3'b011: result_out = a_in | b_in    // Bitwise OR
3'b100: result_out = a_in ^ b_in    // Bitwise XOR
3'b101: result_out = ~a_in         // Bitwise NOT of a_in
3'b110: result_out = a_in << 1     // Left Shift
3'b111: result_out = a_in >> 1     // Right Shift
default: result_out = 10'b0          // Default
```

The design operates on the positive edge of the clock and does not include a reset signal.

2. Output:

The design produces a 9-bit output value (`result_out`), determine by the operation selected through `op_in`.

- **Testbench architecture components:**

- a. Interface: (name – alu_if.sv)

The interface facilitates communication between the DUT (design under test) and the testbench. It consists of logic type inputs and output. The clocking blocks (`drv_cb` & `mon_cb`) for driver and monitor provides the system clock synchronization to the signals. Additionally, modports (`MON_MP` & `DRV_MP`) determine the direction of the signals.

- b. Transaction: (name – alu_trans.sv)

A **transaction class** is like a container that holds all the information about a single transaction in a verification environment.

A transaction class is a structured way to group signals together, store their values, and provide a method to show those values when needed.

1. Class Properties:

- rand logic [7:0] `a_in`, `b_in`
- rand logic [2:0] `op_in`
- logic [8:0] `result_out`
- static int `trans_count`

2. Class Methods:

- `display()`
- `post_randomize()` : to increment the transaction count

- c. Generator: (name – alu_gen.sv)

A generator class contains transaction and mailbox (`gen2drv`) properties. The constructor gets argument as a mailbox. The start method randomize the transaction packet and put it in the mailbox.

1. Class properties:

- `alu_trans trans_h;`

- mailbox #(alu_trans) gen2drv;

2. Class methods:

- start()
- new()

d. Driver: (name – alu_drv.sv)

The alu_drv class is performs the drive operation from alu_gen to the dut interface. It contains the transaction mailbox and virtual interface as properties.

1. Class properties

- alu_trans trans_h
- mailbox #(alu_trans) gen2drv;

2. Class methods

- new()
- drive()
- start() : is to start driving the signals

e. Monitor: (name – alu_mon.sv)

Monitor class contains the trans type property and the interface to get the signals from the dut. The role of the class is to monitor the signals from dut and send it to the scoreboard and reference model.

1. Class Properties

- alu_trans trans_h;
- mailbox #(alu_trans) mon2rm;
- mailbox #(alu_trans) mon2sb;

2. Class methods

- new()
- monitor() : virtual task monitor is to sample the dut outputs
- start()

f. Reference Model: (name – alu_rm.sv)

Class alu_rm is performs the same logic as dut on the input signals and compare it to the dut output. And send the result to the scoreboard to keep record.

1. Class properties

- alu_trans alu_trans
- mailbox #(alu_trans) rm2sb;
- mailbox #(alu_trans) mon2rm;

2. Class method

- new();
- alu_ref_model(); function returns the logic value after performing necessary operation.
- Ref_result(); virtual task to store the result into class object.
- Start();

g. Scoreboard: (name – alu_sb.sv)

Class to perform comparison and records the result and display the report.

1. Class properties:

- alu_trans trans_mon_h
- alu_trans trans_ref_h
- alu_trans cov_data

- static int arith_trans_count: to count the arithmetic transactions.
- static int logic_trans_count: to count the logic transactions.
- static int shift_trans_count: to count the shift transactions.

2. Class methods

- new();
- compare();
- start();
- count_ops();
- report();

The function coverage is declared inside the alu_sb class itself.

```
covergroup alu_cov;
  option.per_instance = 1;
```

```
A_IN: coverpoint cov_data.a_in {
  bins ZERO = {0};
  bins LOW1 = {[1 : 25]};
  bins LOW2 = {[26 : 50]};
  bins MID_LOW = {[51 : 75]};
  bins MID = {[76 : 125]};
  bins MID_HIGH = {[126 : 150]};
  bins HIGH1 = {[151 : 200]};
  bins HIGH2 = {[201 : 224]};
  bins MAX = {225};
}
```

```
B_IN: coverpoint cov_data.b_in {
  bins ZERO = {0};
  bins LOW1 = {[1 : 25]};
  bins LOW2 = {[26 : 50]};
  bins MID_LOW = {[51 : 75]};
  bins MID = {[76 : 125]};
  bins MID_HIGH = {[126 : 150]};
  bins HIGH1 = {[151 : 200]};
  bins HIGH2 = {[201 : 224]};
```

```

bins MAX = {225};

}

OP_IN: coverpoint cov_data.op_in {
    bins ADD = {3'b000};
    bins SUB = {3'b001};
    bins AND = {3'b010};
    bins OR = {3'b011};
    bins XOR = {3'b100};
    bins NOT = {3'b101};
    bins SHL = {3'b110};
    bins SHR = {3'b111};
}

cross A_IN, B_IN, OP_IN;

```

endgroup : alu_cov

h. Package: (name – alu_pkg.sv)

The package inbuild class contains all the files required for the testbench and top.

1. class properties: global int no_of_transactions : to keep the count of transactions.

i. Test: (name – alu_test.sv)

Test class builds the environment and connect interface to dut and env. The assignment of the alu_trans handle is also declared inside the test class.

j. Top: (name – alu_top.sv)

Top class connects the dut to test and generates the clock signal. It performs the testcases depending upon base and derived class.

In this svtb, the regression method is used to perform five different testcases to achieve 100% functional coverage for the input coverpoints.