Design and Simulation of ALU & Verification Methods

ALU Verilog Design

An Arithmetic Logic Unit (ALU) is a digital circuit that performs arithmetic and logical operations. In Verilog, an ALU can be designed as a module that takes inputs from the control unit and operands and generates output based on the type of operation performed. The ALU module typically includes different sub-modules such as adders, subtractors, and logic gates to perform specific operations.
A simple ALU Verilog design can include the following modules:

- A control unit that provides operation control signals to the ALU
- Adder and subtractor modules for arithmetic operations
- Logic gates such as AND, OR, and XOR gates for logical operations

Specification of the ALU of this assignment has been provided. The specification has been mentioned below-

- Inputs are 32 Bits [Parameterizing is preferred]
- Output is a 64 bits signal
- Operations

Opcode	Operation
00001	Add
00010	Subtract
00011	Multiply
00100	1-bit Right Shift
00101	1-bit Left Shift
00110	Bit-wise AND
00111	Bit-wise OR
01000	Adding 1 to Input A
01010	Subtracting 1 from Input A

Block diagram of the ALU is-

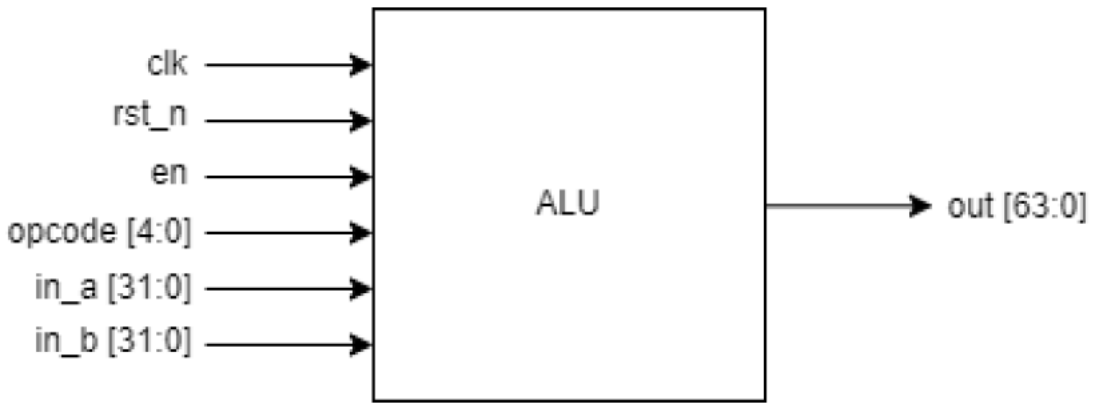

Figure 1: Block Diagram of ALU

Verilog Code of ALU-

```
module alu (out,a,b,op_code,clk,rst,en);
  input clk,rst,en;
  input [31:0]a,b;
```

```
  input [4:0]op_code;
  output [63:0] out;
  reg [63:0] out;

  always@(posedge clk or posedge rst)
  begin
    if (rst == 0) begin
        out = 0;
    end
    else if (en == 1) begin
        case(op_code)
          5'b00001: out=a+b; //8-bit addition
          5'b00010: out=a-b; //8-bit subtraction
          5'b00011: out=a*b; //8-bit multiplication
          5'b00100: out=a<<1; //1-bit left shift
          5'b00101: out=a>>1; //1-bit right shift
          5'b00110: out=a&b; //8-bit bitwise and
          5'b00111: out=a|b; //8-bit bitwise or
          5'b01000: out=a+1; //Adding 1 to Input A
          5'b01010: out=a-1; //Subtracting 1 from Input A
      endcase
    end
  end
endmodule
```

Direct Testbench

A direct test bench is a simple Verilog test bench that provides input values to the DUT (Device Under Test) and checks the output values for correctness. In the case of an ALU, the input values would be the two operands and the control signal, and the output value would be the result of the operation. The direct test bench is easy to write, but it can be time-consuming to manually generate input values and expected output values for a large number of test cases.

Verilog Code of Direct Testbench-

```
`timescale 1ns/1ns
module testbench;
  reg clk,rst,en;
  reg [31:0] a,b;
  reg [4:0] op_code;
  wire [63:0] out;

  // Clock generation
  initial
    forever #5 clk = ~clk;

  // DUT instantiation
  alu DUT( .out (out), .a (a), .b (b), .op_code (op_code), .clk (clk), .rst(rst),
.en(en));
```

```
initial begin
  #0
    clk = 1'b0;
    op_code = 5'bx;
    en = 0;
    rst = 0;
  #5
    rst =1;
    en=1;

  #10
    op_code = 5'b00001;
    a = 8'h00000019;
    b = 8'h00000014;
    //out = 0000002D
  #10
    op_code = 5'b00010;
    a = 8'h075A2988;
    b = 8'h0023CAB9;
  //out = 07365ECF
  #10
    op_code = 5'b00011;
    a = 8'h3B9AC9FF;
    b = 8'h34FB5E38;
  //out = 0C55F7BBB90CD220
  #10
    op_code = 5'b00100;
    a = 32'b00110100111111011010111000111000;

  #10
    op_code = 5'b00101;
    a = 32'b00110100111111011010111000111000;

  #10
    op_code = 5'b00110;
    a = 32'b00110100111111011010111000111000;
    b = 32'b00000100101000101100101101110001;
  //out 00000100101000100100101000110000

  #10
    op_code = 5'b00111;
    a = 32'b00110100111111011010111000111000;
    b = 32'b00000100101000101100101101110001;
      //out 00110100111111011110111101111001
  #10
    op_code = 5'b01000;
    a = 8'h04A2CB71;

  #10
    op_code = 5'b01010;
    a = 8'h04A2CB75;
```

```
    $finish;
  end
  //generating dump file
  initial begin
    $dumpfile("aluDirectDump.vcd");
    $dumpvars;
  end
endmodule
```

Simulation Result-

All the simulation of this assignment has been done on the EDA Playground. The waveform output in hexadecimal base has been shown below-

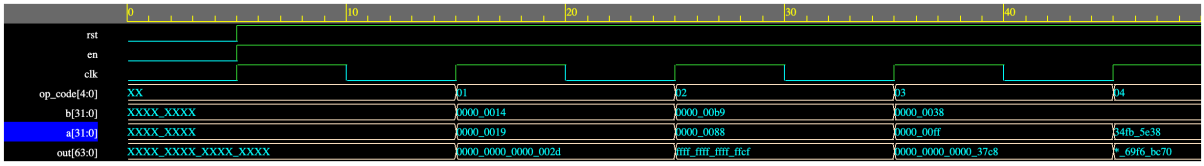

Figure 2: Simulation Result in HEX for opcode 1~4

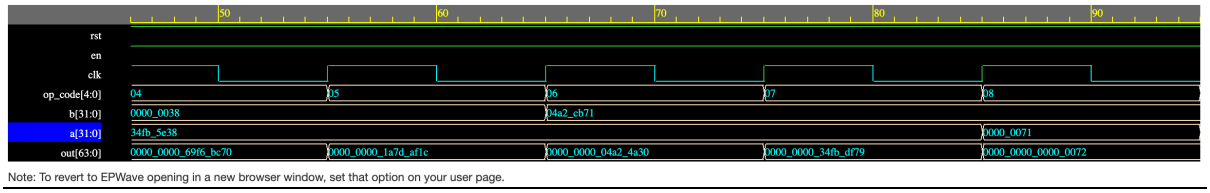

Figure 3: Simulation Result in HEX for opcode 5~8 & 10

Waveform output in binary base for opcode 00100,00101,00110 & 00111-

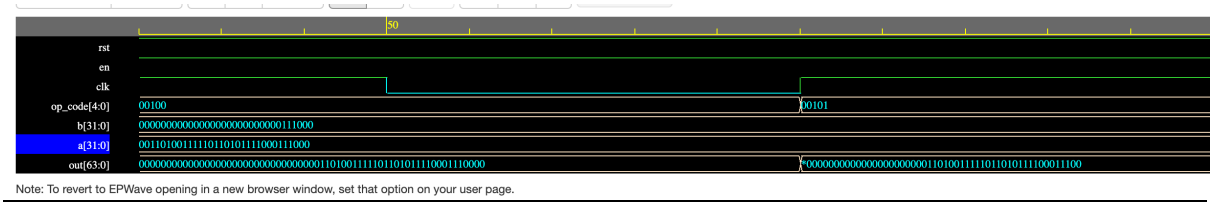

Figure 4: Simulation Result in BIN for opcode 4 & 5

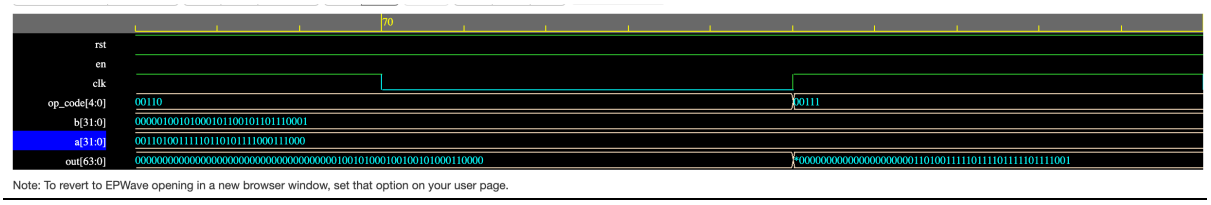

Figure 5: Simulation Result in BIN for opcode 6 & 7

Flat Testbench

A flat test bench is a Verilog test bench that instantiates the design under test (DUT) and provides input stimuli to the DUT. The flat test bench is simple and easy to implement, but it

may not be suitable for complex designs. In a flat test bench, the input stimuli are typically hard-coded in the Verilog code, and the output responses are manually checked.
A simple flat test bench for the ALU design can include the following steps:

- Instantiate the ALU module
- Provide input stimuli to the ALU module
- Wait for the output response from the ALU module
- Check the output response against the expected result

Verilog Code of Direct Testbench-

```
`timescale 1ns/1ns
module testbench;
  reg clk,rst,en;
  reg [31:0] a,b;
  reg [4:0] op_code;
  wire [63:0] out;

  // Clock generation
  initial
    forever #5 clk = ~clk;

  // Module instantiation
  alu DUT( .out (out), .a (a), .b (b), .op_code (op_code), .clk (clk), .rst(rst),
.en(en));

  initial begin
    #0
      clk = 1'b0;
      op_code = 5'bx;
      en = 1;
      rst = 0;
    #5 rst =1;
    #10
      op_code = 5'b00001;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b00010;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b00011;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b00100;
```

```
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b00101;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b00110;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b00111;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b01000;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    #10
      op_code = 5'b01010;
      a = $random;
      b = $random;
    #10
      result_checker(op_code,a,b,out);
    $finish;
  end

  // Task : result_checker
  task result_checker(input [4:0] op_code,input [31:0] a,b,input
[63:0]resulted_out);
    begin
      reg [63:0] out;
      if(op_code==5'b00001)  out=a+b;
      else if(op_code==5'b00010) out=a-b;
      else if(op_code==5'b00011) out=a*b;
      else if(op_code==5'b00100) out=a<<1;
      else if(op_code==5'b00101) out=a>>1;
      else if(op_code==5'b00110) out=a&b;
      else if(op_code==5'b00111) out=a|b;
      else if(op_code==5'b01000) out=a+1;
      else if(op_code==5'b01010) out=a-1;
```

```
      if(resulted_out == out)
        $display("Passed : a=%d, b=%d, s=%d, resulted_out=%d, expected_out=%d",a
,b,op_code,resulted_out,out);
      else
        $display("Failed : a=%d, b=%d, s=%d, resulted_out=%d, expected_out=%d",
a,b,op_code,resulted_out,out);
      end
  endtask


  //generating dump file
  initial begin
    $dumpfile("aluFlatDump.vcd");
    $dumpvars;
  end
endmodule
```

Simulation Result-

```
Passed : a= 303379748, b=3230228097, s= 1, resulted_out=          3533607845, expected_out=          3533607845
Passed : a=2223298057, b=2985317987, s= 2, resulted_out=18446744072947531686, expected_out=18446744072947531686
Passed : a= 112818957, b=1189058957, s= 3, resulted_out=  134148391340247849, expected_out=  134148391340247849
Passed : a=2999092325, b=2302104082, s= 4, resulted_out=          5998184650, expected_out=          5998184650
Passed : a=  15983361, b= 114806029, s= 5, resulted_out=             7991680, expected_out=             7991680
Passed : a= 992211318, b= 512609597, s= 6, resulted_out=           436322612, expected_out=           436322612
Passed : a=1993627629, b=1177417612, s= 7, resulted_out=          1996355565, expected_out=          1996355565
Passed : a=2097015289, b=3812041926, s= 8, resulted_out=          2097015290, expected_out=          2097015290
Passed : a=3807872197, b=3574846122, s=10, resulted_out=          3807872196, expected_out=          3807872196
$finish called from file "testbench.sv", line 79.
$finish at simulation time                  185
```

Figure 6: Simulation Result of Flat Testbench

UVM Testbench

Universal Verification Methodology (UVM) is a popular verification methodology for verifying digital designs. UVM test benches provide a higher level of abstraction and reusability compared to flat test benches. UVM test benches consist of reusable components such as sequences, drivers, monitors, and scoreboards that are connected in a hierarchical structure.

A simple UVM test bench for the ALU design can include the following components:

- A test that defines the input stimuli and expected output responses
- A sequence that generates the input stimuli for the ALU module
- A driver that drives the input stimuli to the ALU module
- A monitor that captures the output responses from the ALU module
- A scoreboard that checks the output responses against the expected result

The UVM test bench provides a higher level of automation and verification coverage compared to the flat test bench. It allows for a more efficient verification process and faster time-to-market for the design.

Module parts of this testbench are-

1. Top module

2. Interface
3. DUT
4. Test
5. Environment
6. Agent
7. Sequence Item
8. Base Sequence
9. Test Sequence
10. Sequencer
11. Driver
12. Monitor
13. Scoreboard

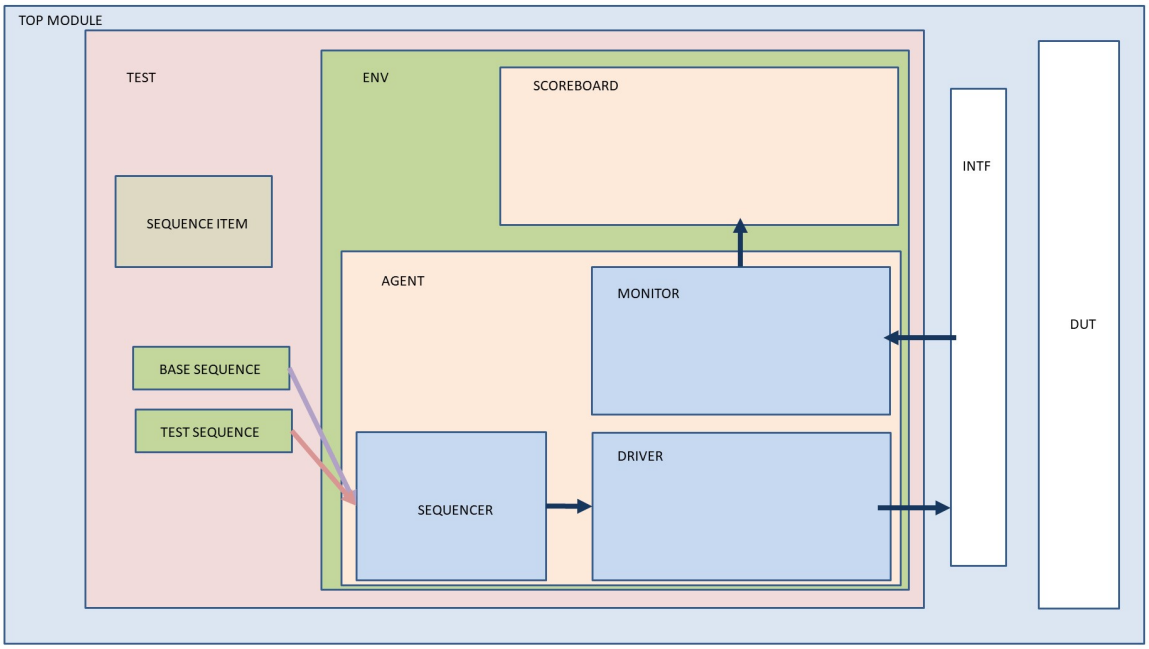

Figure 7: Modular Representation of the UVM based Verification

Verilog code of each module are given below-

#1 Top Module-

```
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "interface.sv"
`include "sequence_item.sv"
`include "sequence.sv"
`include "sequencer.sv"
`include "driver.sv"
`include "monitor.sv"
`include "agent.sv"
`include "scoreboard.sv"
`include "env.sv"
`include "test.sv"
```

```
module testbench;
  logic clk;
  alu_if intf(.clk(clk));

  //DUT called
  alu dut(
    .out(intf.result),
    .a(intf.a),
    .b(intf.b),
    .op_code(intf.op_code),
    .clk(intf.clk),
    .rst(intf.rst),
    .en(intf.en));
  //interface handling
  initial begin
    uvm_config_db #(virtual alu_if)::set(null, "*", "vif", intf );
  end
  //initiate test
  initial begin
    run_test("alu_test");
  end
  //setting clock
  initial begin
    clk=0;
    #5;
    forever begin
      clk=~clk;
      #2;
    end
  end
  //initiate termination
  initial begin
    #5000;
    $display("Simulation time is over");
    $finish();
  end
  //generating dump file
  initial begin
    $dumpfile("aluUvmDump.vcd");
    $dumpvars;
  end
endmodule
```

#2 Interface-

```
interface alu_if(input logic clk);
  logic rst,en;
  logic [31:0] a,b;
  logic [4:0] op_code;
  logic [63:0] result;
endinterface
```

#3 DUT- here DUT was the Verilog code of ALU block.

#5 Environment-

```
class alu_env extends uvm_env;
  //Utility Macro
  `uvm_component_utils(alu_env)
  alu_agent agent;//environment contains agent class
  alu_scoreboard scb;
  //Constructor
  function new(string name="alu_env", uvm_component parent);
  super.new(name, parent);
    `uvm_info("ENV_Class","Inside Constructor",UVM_HIGH);
  endfunction: new

  //Build component instance
  function void build_phase(uvm_phase phase);
  super.build_phase(phase);
    agent = alu_agent::type_id::create("agent", this);
    scb = alu_scoreboard::type_id::create("scb", this);
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  agent.mon.monitor_port.connect(scb.scoreboard_port);
  endfunction: connect_phase

endclass: alu_env
```

#6 Agent-

```
class alu_agent extends uvm_agent;
  //Utility Macro
  `uvm_component_utils(alu_agent)
  //agent contains driver, monitor and sequencer class
  alu_driver drv;
  alu_monitor mon;
  alu_sequencer seqr;

  //Constructor
  function new(string name="alu_agent", uvm_component parent);
  super.new(name, parent);
    `uvm_info("AGENT_Class","Inside Constructor",UVM_HIGH);
  endfunction: new

  //Build component instance
  function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  drv = alu_driver::type_id::create("drv", this);
    mon = alu_monitor::type_id::create("mon", this);
    seqr = alu_sequencer::type_id::create("seqr", this);
  endfunction: build_phase
endclass: alu_agent
```

#7 Sequence Item-

```
class alu_sequence_item extends uvm_sequence_item;
  `uvm_object_utils(alu_sequence_item)

  //Instatiation
  rand logic rst;
  //rand logic en;
  logic en=1;
  rand logic [31:0] a,b;
  rand logic [4:0] op_code;
  logic [63:0] result;

  //Default Constrain
  constraint input1_c {a inside {[10000:20000]};}
  constraint input2_c {b inside {[1:10000]};}
  constraint op_code_c {op_code inside {1,2,3,4,5,6,7,8,10};}

  //Constructor
  function new(string name="alu_sequence_item");
  super.new(name);
    `uvm_info("SEQ_ITEM_Class","Inside Constructor",UVM_HIGH);
  endfunction: new
endclass: alu_sequence_item
```

#8 Base Sequence

```
class alu_base_sequence extends uvm_sequence#(alu_sequence_item);
  //Utility Macro
  `uvm_object_utils(alu_base_sequence)
  alu_sequence_item rst_pkt;

  //Constructor
  function new(string name="alu_base_sequence");
  super.new(name);
    `uvm_info("SEQuence_Class","Inside Constructor",UVM_HIGH);
  endfunction: new

  //Create transaction, send transaction to driver for #Run of times
  task body();
    `uvm_info("SEQuence_Class","Inside Body",UVM_HIGH);
    rst_pkt = alu_sequence_item::type_id::create("rst_pkt");
    start_item(rst_pkt);
    rst_pkt.randomize() with {rst==0;};
    finish_item(rst_pkt);
  endtask: body
endclass: alu_base_sequence
```

#9 Test Sequence-

```
class alu_test_sequence extends alu_base_sequence;
  `uvm_object_utils(alu_test_sequence)
  alu_sequence_item item;

  //Constructor
```

```
  function new(string name= "alu_test_sequence");
    super.new(name);
    `uvm_info("TEST_SEQ", "Inside Constructor!", UVM_HIGH)
  endfunction

  //Body Task
  task body();
    `uvm_info("TEST_SEQ", "Inside body task!", UVM_HIGH)
    item = alu_sequence_item::type_id::create("item");
    start_item(item);
    item.randomize() with {rst==1;};
    finish_item(item);
  endtask: body
endclass: alu_test_sequence
```

#10 Sequencer-

```
class alu_sequencer extends uvm_sequencer#(alu_sequence_item);
  //Utility Macro
  `uvm_component_utils(alu_sequencer)

  //Constructor
  function new(string name="alu_sequencer", uvm_component parent);
  super.new(name, parent);
    `uvm_info("SEQ_Class","Inside Constructor",UVM_HIGH);
  endfunction: new

  //Build component instance
  function void build_phase(uvm_phase phase);
  super.build_phase(phase);
    `uvm_info("SEQ_Class","Inside Build Phase",UVM_HIGH);
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
       super.connect_phase(phase);
  endfunction: connect_phase
endclass: alu_sequencer
```

#11 Driver-

```
class alu_driver extends uvm_driver#(alu_sequence_item);
  //Utility Macro
  `uvm_component_utils(alu_driver)
  virtual alu_if vif;
  alu_sequence_item item;

  //Constructor
  function new(string name="alu_driver", uvm_component parent);
  super.new(name, parent);
    `uvm_info("DRIVER_Class","Inside Constructor",UVM_HIGH);
  endfunction: new
```

```
  //Build component instance
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if(!(uvm_config_db #(virtual alu_if)::get(this, "*", "vif", vif))) begin
      `uvm_error("DRIVER_CLASS", "Failed to get VIF from config DB!")
    end
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
       super.connect_phase(phase);
  endfunction: connect_phase

   //Run sequence
  task run_phase(uvm_phase phase);
    `uvm_info("Driver_Class", "In Run Phase", UVM_HIGH)
    forever begin
      item = alu_sequence_item::type_id::create("item");
      seq_item_port.get_next_item(item);
      drive(item);
      seq_item_port.item_done();
    end
  endtask: run_phase

  //Defining drive task
  task drive(alu_sequence_item item);
    @(posedge vif.clk);
    vif.en <= item.en;
    vif.rst <= item.rst;
    vif.a <= item.a;
    vif.b <= item.b;
    vif.op_code <= item.op_code;
  endtask: drive
endclass: alu_driver
```

#12 Monitor-

```
class alu_monitor extends uvm_monitor;
  //Utility Macro
  `uvm_component_utils(alu_monitor)
  virtual alu_if vif;
  alu_sequence_item item;
  uvm_analysis_port #(alu_sequence_item) monitor_port;

  //Constructor
  function new(string name="alu_monitor", uvm_component parent);
  super.new(name, parent);
    `uvm_info("MONITOR_Class","Inside Constructor",UVM_HIGH);
  endfunction: new

  //Build component instance
```

```
  function void build_phase(uvm_phase phase);
  super.build_phase(phase);
    monitor_port = new("monitor_port", this);
    if(!(uvm_config_db #(virtual alu_if)::get(this, "*", "vif", vif))) begin
      `uvm_error("DRIVER_CLASS", "Failed to get VIF from config DB!")
    end
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
       super.connect_phase(phase);
  endfunction: connect_phase

   //Run sequence
  task run_phase(uvm_phase phase);
    `uvm_info("MONITOR_Class","Inside Run Phase",UVM_HIGH);
    forever begin
      item = alu_sequence_item::type_id::create("item");
      wait(vif.rst);

      //input interconnection
      @(posedge vif.clk);
      item.en = vif.en;
      item.a = vif.a;
      item.b = vif.b;
      item.op_code = vif.op_code;

      //output interconnection
      @(posedge vif.clk);
      item.result = vif.result;

      monitor_port.write(item);
    end
  endtask: run_phase
endclass: alu_monitor
```

#13 Scoreboard-

```
class alu_scoreboard extends uvm_test;
  `uvm_component_utils(alu_scoreboard)
  uvm_analysis_imp #(alu_sequence_item, alu_scoreboard) scoreboard_port;
  alu_sequence_item transactions[$];

  //Constructor
  function new(string name = "alu_scoreboard", uvm_component parent);
    super.new(name, parent);
    `uvm_info("SCB_CLASS", "Inside Constructor!", UVM_HIGH);
  endfunction: new
  //Build Phase
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info("SCB_CLASS", "Build Phase!", UVM_HIGH);
```

```
    scoreboard_port = new("scoreboard_port", this);
  endfunction: build_phase
  //Connect Phase
  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    `uvm_info("SCB_CLASS", "Connect Phase!", UVM_HIGH);
  endfunction: connect_phase
  //Write Method
  function void write(alu_sequence_item item);
    transactions.push_back(item);
  endfunction: write
  //Run Phase
  task run_phase (uvm_phase phase);
    super.run_phase(phase);
    `uvm_info("SCB_CLASS", "Run Phase!", UVM_HIGH);

    forever begin
      //get the data and compare it with the result
      alu_sequence_item curr_trans;
      wait((transactions.size() != 0));
      curr_trans = transactions.pop_front();
      compare(curr_trans);
    end
  endtask: run_phase

  //executing compare task
  task compare(alu_sequence_item curr_trans);
    logic [63:0] expected;
    logic [63:0] actual;

    case(curr_trans.op_code)
        5'b00001: expected=curr_trans.a+curr_trans.b;//addition
        5'b00010: expected= curr_trans.a - curr_trans.b; //subtraction
        5'b00011: expected=curr_trans.a * curr_trans.b; //multiplication
        5'b00100: expected=curr_trans.a<<1; //1-bit left shift
        5'b00101: expected=curr_trans.a>>1; //1-bit right shift
        5'b00110: expected=curr_trans.a&curr_trans.b; // bitwise and
        5'b00111: expected=curr_trans.a|curr_trans.b; //bitwise or
        5'b01000: expected=curr_trans.a+1; //Adding 1 to Input A
        5'b01010: expected=curr_trans.a-1; //Subtracting 1 from Input A
    endcase
    actual = curr_trans.result;
    if(actual == expected) begin
      `uvm_info("COMPARE", $sformatf("Passed : en=%d, a=%d, b=%d, s=%d,
resulted_out=%d, expected_out=%d",curr_trans.en,curr_trans.a
,curr_trans.b,curr_trans.op_code, actual, expected),UVM_LOW);
    end
    else begin
      `uvm_error("COMPARE", $sformatf("Failed  : en=%d, a=%d, b=%d, s=%d,
resulted_out=%d, expected_out=%d",curr_trans.en,curr_trans.a
,curr_trans.b,curr_trans.op_code, actual, expected));
```

```
    end
  endtask: compare
endclass: alu_scoreboard
```

Verilog Code of Test Module-

```
class alu_test extends uvm_test;
  //Utility Macro
  `uvm_component_utils(alu_test)
  alu_env env; //test holds env module
  alu_base_sequence rst_seq;
  alu_test_sequence test_seq;

  //Constructor
  function new(string name="alu_test", uvm_component parent);
  super.new(name, parent);
    `uvm_info("Test_Class","Inside Constructor",UVM_HIGH);
  endfunction: new

  //Build component instance
  function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  env = alu_env::type_id::create("env", this);
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
       super.connect_phase(phase);
  endfunction: connect_phase

   //Run sequence
  task run_phase(uvm_phase phase);
    `uvm_info("TEST_CLASS", "Run Phase!", UVM_HIGH);
    phase.raise_objection(this);
    //starting reset sequence
    rst_seq = alu_base_sequence::type_id::create("rst_seq");
    rst_seq.start(env.agent.seqr);
    #10;

    repeat(20) begin
      //test_seq
      test_seq = alu_test_sequence::type_id::create("test_seq");
      test_seq.start(env.agent.seqr);
      #10;
    end
    phase.drop_objection(this);
  endtask: run_phase
endclass: alu_test
```

Simulation Result-

```
Running test alu_test...
```

```
25: Passed : en=1, a= 14554, b=2349, s=10, resulted_out= 14553, expected_out= 14553
33: Passed : en=1, a= 14554, b=2349, s=10, resulted_out= 14553, expected_out= 14553
41: Passed : en=1, a= 17336, b=1292, s= 4, resulted_out= 34672, expected_out= 34672
49: Passed : en=1, a= 14209, b=7401, s= 1, resulted_out= 21610, expected_out= 21610
57: Passed : en=1, a= 14209, b=7401, s= 1, resulted_out= 21610, expected_out= 21610
65: Passed : en=1, a= 14512, b=6629, s= 1, resulted_out= 21141, expected_out= 21141
73: Passed : en=1, a= 11972, b=8692, s= 6, resulted_out= 8388, expected_out= 8388
81: Passed : en=1, a= 11972, b=8692, s= 6, resulted_out= 8388, expected_out= 8388
89: Passed : en=1, a= 13084, b=9853, s= 4, resulted_out= 26168, expected_out= 26168
97: Passed : en=1, a= 13641, b=3570, s= 3, resulted_out= 48698370, expected_out=
48698370
105: Passed : en=1, a= 13641, b=3570, s= 3, resulted_out= 48698370, expected_out=
48698370
113: Passed : en=1, a= 15364, b=4021, s= 6, resulted_out= 3076, expected_out= 3076
121: Passed : en=1, a= 14207, b=2328, s= 6, resulted_out= 280, expected_out= 280
129: Passed : en=1, a= 14207, b=2328, s= 6, resulted_out= 280, expected_out= 280
137: Passed : en=1, a= 10016, b=3754, s=10, resulted_out= 10015, expected_out=
10015
145: Passed : en=1, a= 13331, b=5917, s= 8, resulted_out= 13332, expected_out=
13332
153: Passed : en=1, a= 13331, b=5917, s= 8, resulted_out= 13332, expected_out=
13332
161: Passed : en=1, a= 16669, b=2197, s= 1, resulted_out= 18866, expected_out=
18866
169: Passed : en=1, a= 18825, b=6630, s= 2, resulted_out= 12195, expected_out=
12195
177: Passed : en=1, a= 18825, b=6630, s= 2, resulted_out= 12195, expected_out=
12195
185: Passed : en=1, a= 17356, b= 610, s= 5, resulted_out= 8678, expected_out= 8678
193: Passed : en=1, a= 13076, b=3114, s= 7, resulted_out= 16190, expected_out=
16190
201: Passed : en=1, a= 13076, b=3114, s= 7, resulted_out= 16190, expected_out=
16190
209: Passed : en=1, a= 16227, b=2974, s= 7, resulted_out= 16383, expected_out=
16383
217: Passed : en=1, a= 11448, b=3973, s= 1, resulted_out= 15421, expected_out=
15421
225: Passed : en=1, a= 11448, b=3973, s= 1, resulted_out= 15421, expected_out=
15421
233: Passed : en=1, a= 10327, b=2132, s= 7, resulted_out= 10327, expected_out=
10327
241: Passed : en=1, a= 18387, b=5285, s= 3, resulted_out= 97175295, expected_out=
97175295
249: Passed : en=1, a= 18387, b=5285, s= 3, resulted_out= 97175295, expected_out=
97175295

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   32
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[COMPARE]    29
[RNTST]     1
[TEST_DONE]     1
[UVM/RELNOTES]     1
```