**design.sv**

// Code your design here

module alu (out,a,b,s,clk);

input clk;

input [8:0]a,b;

input [3:0]s;

output [8:0]out;

reg [8:0]out;

//,flag;

always@(posedge clk) begin

case(s)

4'b0000: out=a+b; //8-bit addition

4'b0001: out=a-b; //8-bit subtraction

4'b0010: out=a\*b; //8-bit multiplication

4'b0011: out=a/b; //8-bit division

4'b0100: out=a%b; //8-bit modulo division

4'b0101: out=a&&b; //8-bit logical and

4'b0110: out=a||b; //8-bit logical or

4'b0111: out=!a; //8-bit logical negation

4'b1000: out=~a; //8-bit bitwise negation

4'b1001: out=a&b; //8-bit bitwise and

4'b1010: out=a|b; //8-bit bitwise or

4'b1011: out=a^b; //8-bit bitwise xor

4'b1100: out=a<<1; //left shift

4'b1101: out=a>>1; //right shift

4'b1110: out=a+1; //increment

4'b1111: out=a-1; //decrement

endcase

end

endmodule

**testbench.sv**

`include "uvm\_macros.svh"

import uvm\_pkg::\*;

`include "alu\_if.sv"

`include "alu\_sequencer.sv"

`include "alu\_monitor.sv"

`include "alu\_driver.sv"

`include "alu\_agent.sv"

`include "alu\_coverage.sv"

`include "alu\_scoreboard.sv"

`include "alu\_env.sv"

`include "alu\_random\_test.sv"

`include "alu\_addition\_test.sv"

//`include "alu\_multiplication\_test.sv"

module testbench;

// Set run count

int count = 25;

// Generate Clock

bit clk;

always #5 clk = ~clk;

//Interface declaration

alu\_if vif(clk);

//DUT instance, interface signals are connected to the DUT ports

alu DUT (

.a(vif.a),

.b(vif.b),

.s(vif.s),

.out(vif.out),

.clk(clk)

);

initial begin

//Registers the Interface in the configuration block so that other blocks can use it

uvm\_resource\_db#(virtual alu\_if)::set(.scope("\*"), .name("alu\_if"), .val(vif));

//Registers the run count in the configuration block so that other blocks can use it

//uvm\_resource\_db#(int)::set(.scope("\*"), .name("count"), .val(count));

uvm\_config\_db#(int)::set(uvm\_root::get(), "\*", "count", count);

//Executes the test

run\_test("alu\_random\_test");

end

initial begin

$dumpfile("dump.vcd");

$dumpvars;

end

endmodule

**alu\_agent.sv**

class alu\_agent extends uvm\_agent;

`uvm\_component\_utils(alu\_agent)

// Analysis Port

uvm\_analysis\_port#(alu\_transaction) agent\_ap;

// Component Instances

alu\_sequencer alu\_seq;

alu\_driver alu\_drv;

alu\_monitor alu\_mon;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name, uvm\_component parent);

super.new(name, parent);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

agent\_ap = new(.name("agent\_ap"), .parent(this));

if(is\_active == UVM\_ACTIVE) begin

alu\_seq = alu\_sequencer::type\_id::create(.name("alu\_seq"), .parent(this));

alu\_drv = alu\_driver::type\_id::create(.name("alu\_drv"), .parent(this));

end

alu\_mon = alu\_monitor::type\_id::create(.name("alu\_mon"), .parent(this));

endfunction: build\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* connect\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

if(is\_active == UVM\_ACTIVE) begin

alu\_drv.seq\_item\_port.connect(alu\_seq.seq\_item\_export);

end

alu\_mon.mon\_ap.connect(agent\_ap);

endfunction: connect\_phase

endclass: alu\_agent

**alu\_assertions.sv**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Assertions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

property addition;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 0, op\_a = a, op\_b = b) |=> (out == (op\_a+op\_b));

endproperty

assert property (addition);

//else $display("Failed");

// cover property (addition);

property subtraction;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 1, op\_a = a, op\_b = b) |=> out == (op\_a - op\_b);

endproperty

assert property (subtraction);

property multiplication;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 2, op\_a = a, op\_b = b) |=> out == (op\_a \* op\_b);

endproperty

assert property (multiplication);

property division;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 3, op\_a = a, op\_b = b) |=> out == (op\_a / op\_b);

endproperty

assert property (division);

property modulo\_division;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 4, op\_a = a, op\_b = b) |=> out == (op\_a % op\_b);

endproperty

assert property (modulo\_division);

property logical\_and;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 5, op\_a = a, op\_b = b) |=> (out == (op\_a && op\_b));

endproperty

assert property (logical\_and);

//else $display("a=%d, b=%d, out=%d",a,b,out);

property logical\_or;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 6, op\_a = a, op\_b = b) |=> out == (op\_a || op\_b);

endproperty

assert property (logical\_or);

property logical\_negation;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 7, op\_a = a, op\_b = b) |=> out == !op\_a;

endproperty

assert property (logical\_negation);

property bitwise\_negation;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 8, op\_a = a, op\_b = b) |=> out == ~op\_a;

endproperty

assert property (bitwise\_negation);

property bitwise\_and;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 9, op\_a = a, op\_b = b) |=> out == (op\_a & op\_b);

endproperty

assert property (bitwise\_and);

property bitwise\_or;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 10, op\_a = a, op\_b = b) |=> out == (op\_a | op\_b);

endproperty

assert property (bitwise\_or);

property bitwise\_xor;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 11, op\_a = a, op\_b = b) |=> out == (op\_a ^ op\_b);

endproperty

assert property (bitwise\_xor);

property left\_shift;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 12, op\_a = a, op\_b = b) |=> out == op\_a << 1;

endproperty

assert property (left\_shift);

property right\_shift;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 13, op\_a = a, op\_b = b) |=> out == op\_a >> 1;

endproperty

assert property (right\_shift);

property increment;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 14, op\_a = a, op\_b = b) |=> out == op\_a + 1;

endproperty

assert property (increment);

property decrement;

logic [8:0] op\_a, op\_b;

@(posedge clk) (s == 15, op\_a = a, op\_b = b) |=> out == op\_a - 1;

endproperty

assert property (decrement);

**alu\_converge.sv**

class alu\_coverage extends uvm\_subscriber#(alu\_transaction);

`uvm\_component\_utils(alu\_coverage)

alu\_transaction alu\_tx\_cg;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Covergroup \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

covergroup alu\_cg;

option.per\_instance = 1;

selector: coverpoint alu\_tx\_cg.s {

bins addition = {0};

bins subtraction = {1};

bins multiplication = {2};

bins division = {3};

bins modulo\_division = {4};

bins logical\_and = {5};

bins logical\_or = {6};

bins logical\_negation = {7};

bins bitwise\_negation = {8};

bins bitwise\_and = {9};

bins bitwise\_or = {10};

bins bitwise\_xor = {11};

bins left\_shift = {12};

bins right\_shift = {13};

bins increment = {14};

bins decrement = {15};

option.at\_least = 1;

}

op\_a: coverpoint alu\_tx\_cg.a {

bins zero\_to\_255 = {[0:255]};

}

cross\_a\_s: cross op\_a,selector;

endgroup: alu\_cg

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Assertions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* property addition;

alu\_tx\_cg.s == 0 |=> alu\_tx\_cg.out == alu\_tx\_cg.a+alu\_tx\_cg.b;

endproperty

assert property (addition);

cover property (addition); \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name, uvm\_component parent);

super.new(name, parent);

alu\_cg = new();

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* function - write \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void write(alu\_transaction t);

alu\_tx\_cg = t;

alu\_cg.sample();

endfunction:write

endclass : alu\_coverage

**alu\_driver.sv**

class alu\_driver extends uvm\_driver#(alu\_transaction);

`uvm\_component\_utils(alu\_driver)

// Virtual Interface

virtual alu\_if vif;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name, uvm\_component parent);

super.new(name, parent);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

void'(uvm\_resource\_db#(virtual alu\_if)::read\_by\_name(.scope("\*"), .name("alu\_if"), .val(vif)));

endfunction: build\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* run\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

task run\_phase(uvm\_phase phase);

drive();

endtask: run\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* task - drive() \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

virtual task drive();

alu\_transaction alu\_tx;

//vif.DRIVER.driver\_cb.a <= 0;

//vif.DRIVER.driver\_cb.b <= 0;

//vif.DRIVER.driver\_cb.s <= 0;

forever begin

seq\_item\_port.get\_next\_item(alu\_tx);

// Print the received transaction

//`uvm\_info("alu\_driver", alu\_tx.sprint(), UVM\_LOW);

@(vif.driver\_cb);

vif.DRIVER.driver\_cb.a <= alu\_tx.a;

vif.DRIVER.driver\_cb.b <= alu\_tx.b;

vif.DRIVER.driver\_cb.s <= alu\_tx.s;

seq\_item\_port.item\_done();

end

endtask: drive

endclass:alu\_driver

**alu\_env.sv**

class alu\_env extends uvm\_env;

// Component Instances

alu\_agent agents[];

alu\_scoreboard scb;

alu\_coverage cov;

int num\_agents = 2;

string inst\_name;

`uvm\_component\_utils\_begin(alu\_env)

`uvm\_field\_int(num\_agents, UVM\_ALL\_ON)

`uvm\_component\_utils\_end

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name, uvm\_component parent);

super.new(name, parent);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

virtual function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

if(num\_agents == 0)

`uvm\_fatal("NONUM","'num\_agents' must be set");

agents = new[num\_agents];

for(int i=0; i < num\_agents; i++) begin

$sformat(inst\_name, "agents[%0d]", i);

agents[i] = alu\_agent::type\_id::create(.name(inst\_name), .parent(this));

end

scb = alu\_scoreboard::type\_id::create(.name("scb"), .parent(this));

cov = alu\_coverage::type\_id::create(.name("cov"), .parent(this));

// Set agents ACTIVE/PASSIVE

uvm\_config\_db#(int)::set(uvm\_root::get(),"\*.agents[0]","is\_active", UVM\_ACTIVE);

uvm\_config\_db#(int)::set(uvm\_root::get(),"\*.agents[1]","is\_active", UVM\_PASSIVE);

endfunction: build\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* connect\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

agents[0].agent\_ap.connect(scb.sb\_export\_before);

agents[1].agent\_ap.connect(scb.sb\_export\_after);

agents[0].agent\_ap.connect(cov.analysis\_export);

endfunction: connect\_phase

endclass: alu\_env

**alu\_if.sv**

interface alu\_if(input clk);

logic [8:0] a,b;

logic [3:0] s;

logic [8:0] out;

clocking driver\_cb @(posedge clk);

default input #1 output #1;

output a,b,s;

endclocking

clocking mon\_cb @(posedge clk);

default input #1 output #2;

input a,b,s;

input out;

endclocking

modport DRIVER (clocking driver\_cb, input clk);

modport MONITOR (clocking mon\_cb, input clk);

`include "alu\_assertions.sv"

endinterface

**alu\_monitor.sv**

class alu\_monitor extends uvm\_monitor;

`uvm\_component\_utils(alu\_monitor)

// Analysis Port

uvm\_analysis\_port#(alu\_transaction) mon\_ap;

// Virtual Interface

virtual alu\_if vif;

alu\_transaction alu\_tx;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name, uvm\_component parent);

super.new(name, parent);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

void'(uvm\_resource\_db#(virtual alu\_if)::read\_by\_name(.scope("\*"), .name("alu\_if"), .val(vif)));

mon\_ap = new(.name("mon\_ap"), .parent(this));

endfunction: build\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* run\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

task run\_phase(uvm\_phase phase);

alu\_tx = alu\_transaction::type\_id::create (.name("alu\_tx"), .contxt(get\_full\_name()));

forever begin

@(vif.mon\_cb);

alu\_tx.a = vif.MONITOR.mon\_cb.a;

alu\_tx.b = vif.MONITOR.mon\_cb.b;

alu\_tx.s = vif.MONITOR.mon\_cb.s;

alu\_tx.out = vif.MONITOR.mon\_cb.out;

// Print the sampled inputs

//`uvm\_info("alu\_monitor", alu\_tx.sprint(), UVM\_LOW);

//Send the transaction to the analysis port

mon\_ap.write(alu\_tx);

end

endtask: run\_phase

endclass: alu\_monitor

**alu\_scoreboard.sv**

//`uvm\_analysis\_imp\_decl(\_before)

//`uvm\_analysis\_imp\_decl(\_after)

class alu\_scoreboard extends uvm\_scoreboard;

`uvm\_component\_utils(alu\_scoreboard)

// Analysis Port

uvm\_analysis\_export #(alu\_transaction) sb\_export\_before;

uvm\_analysis\_export #(alu\_transaction) sb\_export\_after;

// Analysis FIFO

uvm\_tlm\_analysis\_fifo #(alu\_transaction) before\_fifo;

uvm\_tlm\_analysis\_fifo #(alu\_transaction) after\_fifo;

alu\_transaction transaction\_before;

alu\_transaction transaction\_after;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name, uvm\_component parent);

super.new(name, parent);

transaction\_before = new("transaction\_before");

transaction\_after = new("transaction\_after");

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

sb\_export\_before = new("sb\_export\_before", this);

sb\_export\_after = new("sb\_export\_after", this);

before\_fifo = new("before\_fifo", this);

after\_fifo = new("after\_fifo", this);

endfunction: build\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* connect\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void connect\_phase(uvm\_phase phase);

sb\_export\_before.connect(before\_fifo.analysis\_export);

sb\_export\_after.connect(after\_fifo.analysis\_export);

endfunction: connect\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* task - run \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

task run();

forever begin

before\_fifo.get(transaction\_before);

after\_fifo.get(transaction\_after);

//predictor();

compare();

predictor();

end

endtask: run

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* function - Compare \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

virtual function void compare();

if(transaction\_before.out == transaction\_after.out) begin

`uvm\_info(get\_type\_name(),$sformatf("a = %0d, b = %0d, s = %0d, Expected = %0d, Resulted = %0d", transaction\_before.a, transaction\_before.b, transaction\_before.s, transaction\_before.out, transaction\_after.out), UVM\_LOW);

`uvm\_info("PASSED", {"Test: PASSED "}, UVM\_LOW);

end else begin

`uvm\_info(get\_type\_name(),$sformatf("a = %0d, b = %0d, s = %0d, Expected = %0d, Resulted = %0d", transaction\_before.a, transaction\_before.b, transaction\_before.s, transaction\_before.out, transaction\_after.out), UVM\_LOW);

`uvm\_info("FAILED", {"Test: FAILED "}, UVM\_LOW);

end

endfunction: compare

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* function - predictor() \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

virtual function void predictor();

if (transaction\_before.s==0) transaction\_before.out=transaction\_before.a + transaction\_before.b;

else if(transaction\_before.s==1) transaction\_before.out=transaction\_before.a - transaction\_before.b;

else if(transaction\_before.s==2) transaction\_before.out=transaction\_before.a \* transaction\_before.b;

else if(transaction\_before.s==3) transaction\_before.out=transaction\_before.a / transaction\_before.b;

else if(transaction\_before.s==4) transaction\_before.out=transaction\_before.a % transaction\_before.b;

else if(transaction\_before.s==5) transaction\_before.out=transaction\_before.a && transaction\_before.b;

else if(transaction\_before.s==6) transaction\_before.out=transaction\_before.a || transaction\_before.b;

else if(transaction\_before.s==7) transaction\_before.out=!transaction\_before.a;

else if(transaction\_before.s==8) transaction\_before.out=~transaction\_before.a;

else if(transaction\_before.s==9) transaction\_before.out=transaction\_before.a & transaction\_before.b;

else if(transaction\_before.s==10) transaction\_before.out=transaction\_before.a | transaction\_before.b;

else if(transaction\_before.s==11) transaction\_before.out=transaction\_before.a ^ transaction\_before.b;

else if(transaction\_before.s==12) transaction\_before.out=transaction\_before.a << 1;

else if(transaction\_before.s==13) transaction\_before.out=transaction\_before.a >> 1;

else if(transaction\_before.s==14) transaction\_before.out=transaction\_before.a + 1;

else if(transaction\_before.s==15) transaction\_before.out=transaction\_before.a - 1;

endfunction: predictor

endclass: alu\_scoreboard

**alu\_sequence.sv**

class alu\_sequence extends uvm\_sequence#(alu\_transaction);

`uvm\_object\_utils(alu\_sequence)

// Set run count

int count;

// Transaction class handle

rand alu\_transaction alu\_tx;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name = "alu\_sequence");

super.new(name);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* task - body() \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

task body();

// Get the count value from testbench top

//void'(uvm\_resource\_db#(int)::read\_by\_name(.scope(""), .name("count"), .val(count)));

if(!uvm\_config\_db#(int)::get(uvm\_root::get(), "\*", "count", count))

`uvm\_fatal("NOCOUNT", {"count must be set for: ", get\_full\_name(),".count"});

alu\_tx = alu\_transaction::type\_id::create(.name("alu\_tx"), .contxt(get\_full\_name()));

repeat(count) begin

//alu\_tx = alu\_transaction::type\_id::create(.name("alu\_tx"), .contxt(get\_full\_name()));

start\_item(alu\_tx);

assert(this.randomize(alu\_tx));

//`uvm\_info("alu\_sequence", alu\_tx.sprint(), UVM\_LOW);

finish\_item(alu\_tx);

end

endtask: body

endclass: alu\_sequence

**alu\_sequencer.sv**

`include "alu\_transaction.sv"

`include "alu\_sequence.sv"

typedef uvm\_sequencer#(alu\_transaction) alu\_sequencer;

**alu\_transaction.sv**

class alu\_transaction extends uvm\_sequence\_item;

rand bit [8:0] a;

rand bit [8:0] b;

randc bit [3:0] s;

bit [8:0] out;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name = "alu\_transaction");

super.new(name);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Utility and Field macros \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

`uvm\_object\_utils\_begin(alu\_transaction)

`uvm\_field\_int(a,UVM\_ALL\_ON)

`uvm\_field\_int(b,UVM\_ALL\_ON)

`uvm\_field\_int(s,UVM\_ALL\_ON)

`uvm\_field\_int(out,UVM\_ALL\_ON)

`uvm\_object\_utils\_end

/\* constraint inputs {

a inside {[1:10]};

b inside {[1:10]};

s == 0;

}

\*/

endclass: alu\_transaction

**alu\_addition\_test.sv**

class addition\_constraint extends alu\_sequence;

`uvm\_object\_utils(addition\_constraint)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name = "addition\_constraint");

super.new(name);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Constraints \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

constraint inputs {

alu\_tx.a inside {[1:10]};

alu\_tx.b inside {[1:10]};

alu\_tx.s == 0; // Addition

}

endclass: addition\_constraint

//----------------------------------------------------------------------------------//

class alu\_addition\_test extends alu\_random\_test;

`uvm\_component\_utils(alu\_addition\_test)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name = "alu\_addition\_test", uvm\_component parent = null);

super.new(name, parent);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* start\_of\_simulation\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void start\_of\_simulation\_phase(uvm\_phase phase);

set\_type\_override\_by\_type(alu\_sequence::get\_type(), addition\_constraint::get\_type());

endfunction

endclass : alu\_addition\_test

**alu\_multiplication\_test.sv**

class multiplication\_constraint extends alu\_sequence;

`uvm\_object\_utils(multiplication\_constraint)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name = "multiplication\_constraint");

super.new(name);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Constraints \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

constraint inputs {

alu\_tx.a inside {[1:10]};

alu\_tx.b inside {[1:10]};

alu\_tx.s == 2; // Multiplication

}

endclass: multiplication\_constraint

//----------------------------------------------------------------------------------//

class alu\_multiplication\_test extends alu\_random\_test;

`uvm\_component\_utils(alu\_multiplication\_test)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name = "alu\_multiplication\_test", uvm\_component parent = null);

super.new(name, parent);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* start\_of\_simulation\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void start\_of\_simulation\_phase(uvm\_phase phase);

set\_type\_override\_by\_type(alu\_sequence::get\_type(), multiplication\_constraint::get\_type());

endfunction

endclass : alu\_multiplication\_test

**alu\_random\_test.sv**

class alu\_random\_test extends uvm\_test;

`uvm\_component\_utils(alu\_random\_test)

// Component Instances

alu\_env env;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* new - Constructor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name, uvm\_component parent);

super.new(name, parent);

endfunction: new

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

env = alu\_env::type\_id::create(.name("env"), .parent(this));

endfunction: build\_phase

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* run\_phase \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

task run\_phase(uvm\_phase phase);

alu\_sequence seq;

phase.raise\_objection(.obj(this));

seq = alu\_sequence::type\_id::create(.name("seq"), .contxt(get\_full\_name()));

//assert(seq.randomize());

seq.start(env.agents[0].alu\_seq);

phase.drop\_objection(.obj(this));

endtask: run\_phase

endclass: alu\_random\_test