System Verilog

UVM

Assignment 5

By: Magdy Ahmed Abbas Abdelhamid

Question 1

ALSU

Assignment 5 - UVM

This assignment is to practice writing a UVM testbench environment for the ALSU design. Check the notes and classwork codes uploaded on Google classroom to do the assignment. The ALSU design was shared in assignment 3. The assignment will be split into three parts. Use a do file in the 3 parts (I have attached a do file for you to use and modify). Don't forget to import uvm_pkg and uvm_macros in **all** files. **Each class created will be in a separate file and in a package.**

Part #1:

In this part, you will create a top module that will start a UVM test. UVM test will build the UVM environment and then displays a message.

Files to create:

1. top module
2. alsu_test
3. alsu_env

Steps:

1. Create top module where you will instantiate the DUT, interface, assign the interface to the DUT, generate the clock and in an initial block use the global task run_test to run your uvm test environment (alsu_test).
2. Create a class named alsu_test in a separate file in a package, Inside the test class, you will build the environment component (alsu_env) in the build_phase. In the run phase, raise the objection to display a message "Inside the ALSU test" using `uvm_info. Drop the objection after it. Don't forget to import the alsu_env package.
3. Create a class named alsu_env. No phases will be overridden in the environment.

Deliverables:

1. Simulate the top module and make sure that the message is displayed. Take a screenshot to use it in your PDF.

ALSU Design

```
///////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
// Description: ALSU Design
///////////////////////////////////////////////////////////////////////////////
module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;
output reg [15:0] leds;
output reg [5:0] out;
reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2:0] opcode_reg, A_reg, B_reg;
wire invalid_red_op, invalid_opcode, invalid;
assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
assign invalid = invalid_red_op | invalid_opcode;
always @(posedge clk or posedge rst) begin
  if(rst) begin
     cin_reg <= 0;
     red_op_B_reg <= 0;
     red_op_A_reg <= 0;
     bypass_B_reg <= 0;
     bypass_A_reg <= 0;
     direction_reg <= 0;
     serial_in_reg <= 0;
     opcode_reg <= 0;
     A_reg <= 0;
     B_reg <= 0;
  end else begin
     cin_reg <= cin;
     red_op_B_reg <= red_op_B;
     red_op_A_reg <= red_op_A;
     bypass_B_reg <= bypass_B;
     bypass_A_reg <= bypass_A;
     direction_reg <= direction;
     serial_in_reg <= serial_in;
     opcode_reg <= opcode;
     A_reg <= A;
     B_reg <= B;
  end
end
```

```
always @(posedge clk or posedge rst) begin
  if(rst) begin
     leds  <=  0;
  end else begin
      if (invalid)
        leds  <=  ~leds;
      else
        leds  <=  0;
  end
end
always @(posedge clk or posedge rst) begin
  if(rst) begin
    out  <=  0;
  end
  else begin
    if (invalid)
        out  <=  0;
    else if (bypass_A_reg && bypass_B_reg)
      out  <=  (INPUT_PRIORITY  ==  "A")?  A_reg: B_reg;
    else if (bypass_A_reg)
      out  <=  A_reg;
    else if (bypass_B_reg)
      out  <=  B_reg;
    else begin
        case (opcode)
          3'h0: begin
           if (red_op_A_reg && red_op_B_reg)
             out  =  (INPUT_PRIORITY  ==  "A")?  &A_reg: &B_reg;
           else if (red_op_A_reg)
             out  <=  &A_reg;
           else if (red_op_B_reg)
             out  <=  &B_reg;
           else
             out  <=  A_reg & B_reg;
          end
          3'h1: begin
           if (red_op_A_reg && red_op_B_reg)
             out  <=  (INPUT_PRIORITY  ==  "A")?  |A_reg: |B_reg;
           else if (red_op_A_reg)
             out  <=  |A_reg;
           else if (red_op_B_reg)
             out  <=  |B_reg;
           else
             out  <=  A_reg | B_reg;
          end
          3'h2: out  <=  A_reg  +  B_reg;
          3'h3: out  <=  A_reg  *  B_reg;
          3′h4: begin
           if (direction_reg)
             out  <=  {out[4: 0], serial_in_reg};
           else
             out  <=  {serial_in_reg, out[5: 1]};
          end
          3′h5: begin
           if (direction_reg)
             out  <=  {out[4: 0], out[5]};
           else
             out  <=  {out[0], out[5: 1]};
          end
        endcase
    end
  end
end
endmodule
```

Top Module

```
import uvm_pkg: :*;
import alsu_test_pkg: :*;
`include "uvm_macros. svh"
module top ();
bit clk;
initial
begin
clk = 0;
forever
#1 clk = ~clk;
end

alsu_if f1(clk);
ALSU A(f1. A, f1. B, f1. cin, f1. serial_in, f1. red_op_A, f1. red_op_B, f1. opcode,
f1. bypass_A, f1. bypass_B, f1. clk, f1. rst, f1. direction, f1. leds, f1. out);
initial
begin
 run_test("alsu_test");
end
endmodule
```

Interface

```
interface alsu_if (clk);
   input clk;
logic  rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
logic [2: 0] opcode;
logic signed [2: 0] A, B;
logic  [15: 0] leds;
logic  signed [5: 0] out;
endinterface
```

ALSU Test

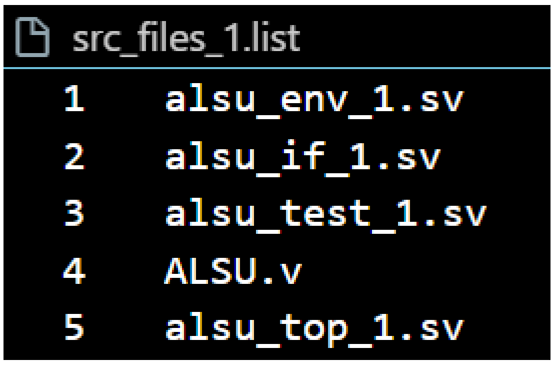
```
package alsu_test_pkg;
import uvm_pkg: :*;
 import alsu_env_pkg: :*;
  `include "uvm_macros. svh"
class alsu_test extends uvm_test;
   `uvm_component_utils(alsu_test)
   alsu_env env;

   function new(string name = "alsu_test", uvm_component parent  = null);
      super. new(name, parent);
   endfunction

   function void build_phase(uvm_phase phase);
      super. build_phase (phase);
      env = alsu_env: : type_id: : create("env", this);
   endfunction

   task run_phase(uvm_phase phase);
      super. run_phase(phase);
      phase. raise_objection(this);
      #100; `uvm_info("run_phase", "inside the ALSU test", UVM_MEDIUM)
      phase. drop_objection(this);
   endtask

endclass

endpackage
```

ALSU Environment

```
package alsu_env_pkg;
   import uvm_pkg::*;

   `include "uvm_macros.svh"
   class alsu_env extends uvm_env;
      `uvm_component_utils(alsu_env)

      function new (string name = "alsu_env", uvm_component parent  = null);
         super.new(name, parent);
      endfunction

function void build_phase(uvm_phase phase);
   super.build_phase(phase);
endfunction

endclass
endpackage
```

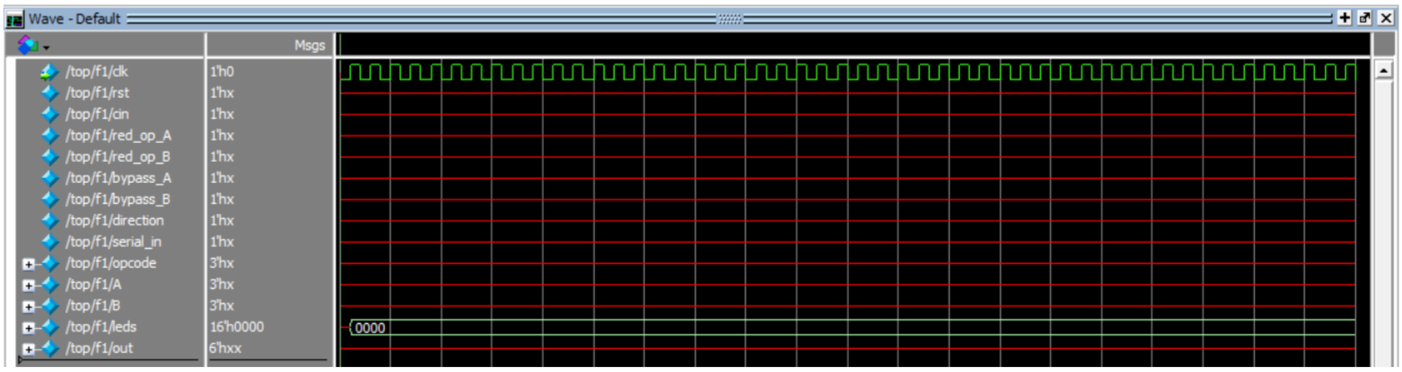
Do File

```
vlib work
vlog -f src_files.list
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all
add wave /top/f1/*
run -all
```


src_files_1.list

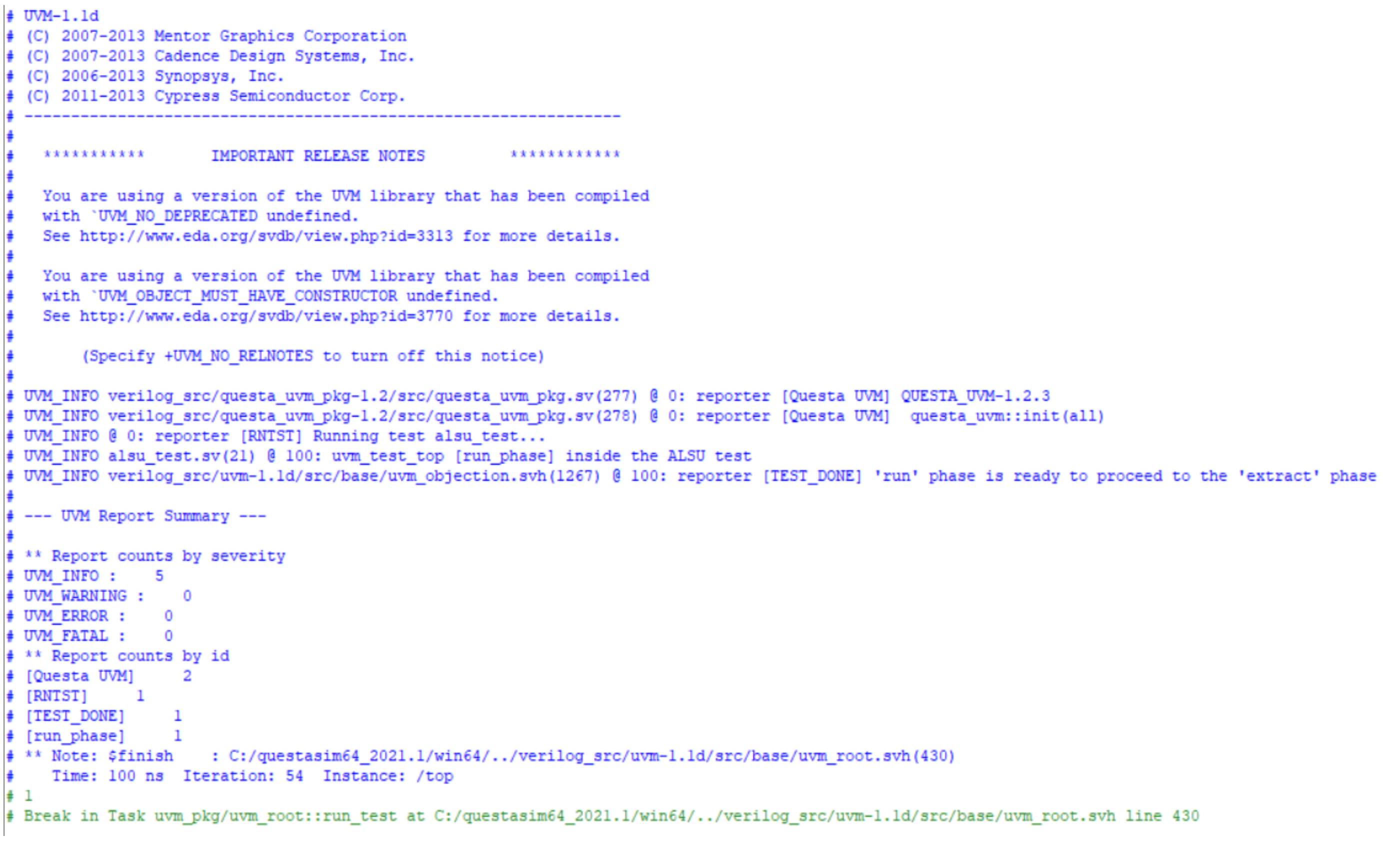
```
alsu_env_1.sv
alsu_if_1.sv
alsu_test_1.sv
ALSU.v
alsu_top_1.sv
```

Waveform



```
/top/f1/clk         1'h0
/top/f1/rst         1'hx
/top/f1/cin         1'hx
/top/f1/red_op_A    1'hx
/top/f1/red_op_B    1'hx
/top/f1/bypass_A    1'hx
/top/f1/bypass_B    1'hx
/top/f1/direction   1'hx
/top/f1/serial_in   1'hx
/top/f1/opcode      3'hx
/top/f1/A           3'hx
/top/f1/B           3'hx
/top/f1/leds        16'h0000
/top/f1/out         6'hxx
```

Transcript



```
# UVM-1.1d
# (C) 2007-2013 Mentor Graphics Corporation
# (C) 2007-2013 Cadence Design Systems, Inc.
# (C) 2006-2013 Synopsys, Inc.
# (C) 2011-2013 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(all)
# UVM_INFO @ 0: reporter [RNTST] Running test alsu_test...
# UVM_INFO alsu_test.sv(21) @ 100: uvm_test_top [run_phase] inside the ALSU test
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 100: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    5
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [run_phase]     1
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 100 ns  Iteration: 54  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

Question 2

Part #2:

In this part, you will pass the virtual interface to the UVM driver to drive the interface.

Files to create:

1. top module
2. alsu_test
3. alsu_env
4. alsu_driver

Steps:

1. Create top module where you will instantiate the DUT, interface, assign the interface to the DUT, generate the clock. Inside the initial block, set the virtual interface in the configuration database (uvm_config_db) using the set method then use the global task run_test to run your uvm test environment (alsu_test).
2. Create a class named alsu_test in a separate file in a package, Inside the test class
   a. Declare a virtual interface named **alsu_test_vif** and handle of the alsu_env
   b. In the build_phase, you will do the following
      i. Retrieve the virtual interface from the configuration database (uvm_config_db) and pass the value of the virtual interface to **alsu_test_vif**
      ii. Set the virtual interface **alsu_test_vif** in the configuration database to all the components under the test class so that any component can retrieve it.
      iii. Build the environment component (alsu_env).
   c. In the run phase, raise the objection, wait for #100 then display a message "Inside the ALSU test" using `uvm_info. Drop the objection after it. Don't forget to import the alsu_env package.
3. Create a class named alsu_env. Inside the build_phase, build the driver (Don't forget to import the driver package)
4. Create a class named alsu_driver extends uvm_driver
   a. Declare a virtual interface named **alsu_driver_vif**
   b. In the build_phase:
      i. Retrieve the virtual interface set by the alsu_test from the configuration database (uvm_config_db) and pass the value of the virtual interface to **alsu_driver_vif**
   c. In the run_phase:
      i. Reset the ALSU, then in a forever loop use $random to randomize the inputs of the ALSU using the virtual interface **alsu_driver_vif.**

Deliverables:

1. Simulate the top module and make sure that the message is displayed. Take a screenshot to use it in your PDF.
2. Add the signals of the interface to wave and make sure that the inputs are driven. Take a screenshot and add it to you PDF.

Top Module

```
import uvm_pkg::*;
import alsu_test_pkg::*;
`include "uvm_macros.svh"
module top ();
bit clk;
initial
begin
clk = 0;
forever
#1 clk = ~clk;
end

alsu_if f1(clk);
ALSU A(f1.A, f1.B, f1.cin, f1.serial_in, f1.red_op_A, f1.red_op_B, f1.opcode, f1.bypass_A,
f1.bypass_B, f1.clk, f1.rst, f1.direction, f1.leds, f1.out);
initial
begin
uvm_config_db#(virtual alsu_if)::set(null, "uvm_test_top", "ALSU_IF", f1);

run_test("alsu_test");
end

endmodule
```

Interface

```
interface alsu_if (clk);
   input clk;
logic  rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
logic [2:0] opcode;
logic signed [2:0] A, B;
logic  [15:0] leds;
logic  signed [5:0] out;
endinterface
```

ALSU Test

```
package alsu_test_pkg;
import uvm_pkg::*;
 import alsu_env_pkg::*;
 `include "uvm_macros.svh"
class alsu_test extends uvm_test;
   `uvm_component_utils(alsu_test)
   alsu_env env;
   virtual alsu_if alsu_test_vif;

   function new(string name = "alsu_test", uvm_component parent  = null);
      super.new(name, parent);
   endfunction

   function void build_phase(uvm_phase phase);
      super.build_phase (phase);
      env = alsu_env::type_id::create("env", this);
   uvm_config_db#(virtual alsu_if)::get(this, "", "ALSU_IF", alsu_test_vif);

   uvm_config_db#(virtual alsu_if)::set(this, " * ", "ALSU_IF_2", alsu_test_vif );

   endfunction

   task run_phase(uvm_phase phase);
      super.run_phase(phase);
      phase.raise_objection(this);
      #100;`uvm_info("run_phase", "inside the ALSU test", UVM_MEDIUM)
      phase.drop_objection(this);
   endtask
endclass

endpackage
```

ALSU Environment

```
package alsu_env_pkg;
   import alsu_driver_pkg::*;
   import uvm_pkg::*;
   `include "uvm_macros.svh"
   class alsu_env extends uvm_env;
      `uvm_component_utils(alsu_env)

alsu_driver driver;

      function new (string name = "alsu_env", uvm_component parent  = null);
         super.new(name, parent);
      endfunction

function void build_phase(uvm_phase phase);
   super.build_phase(phase);
driver = alsu_driver::type_id::create("driver", this);
   endfunction
endclass
endpackage
```

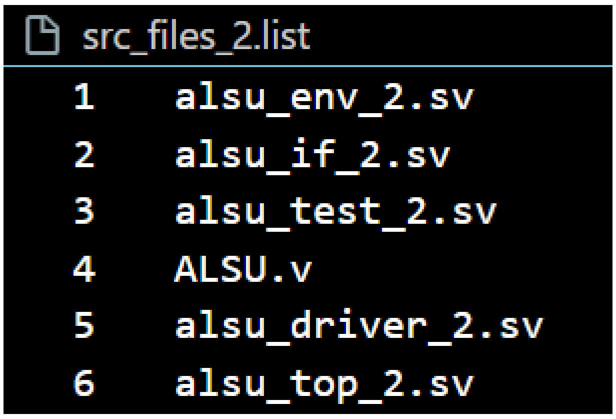
ALSU Driver

```
package alsu_driver_pkg;
   import uvm_pkg::*;
   `include "uvm_macros.svh"

class alsu_driver extends uvm_driver;
   `uvm_component_utils(alsu_driver)
   virtual alsu_if alsu_driver_vif;

   function new(string name = "alsu_driver", uvm_component parent = null);
      super.new(name, parent);
   endfunction

   function void build_phase(uvm_phase phase);
      super.build_phase (phase);
   uvm_config_db #(virtual alsu_if)::get(this, "", "ALSU_IF_2", alsu_driver_vif);
   endfunction
```

```
    task run_phase(uvm_phase phase);
      alsu_driver_vif.rst = 1;
      @(negedge alsu_driver_vif.clk);
      alsu_driver_vif.rst = 0;
      forever
      begin
      @(negedge alsu_driver_vif.clk);
          alsu_driver_vif.A  =  $random;
          alsu_driver_vif.B  =  $random;
          alsu_driver_vif.cin  =  $random;
          alsu_driver_vif.opcode  =  $random;
          alsu_driver_vif.bypass_A  =  $random;
          alsu_driver_vif.bypass_B  =  $random;
          alsu_driver_vif.red_op_A  =  $random;
          alsu_driver_vif.red_op_B  =  $random;
          alsu_driver_vif.direction  =  $random;
          alsu_driver_vif.serial_in  =  $random;
      end
     endtask
    endclass
endpackage
```

Do File

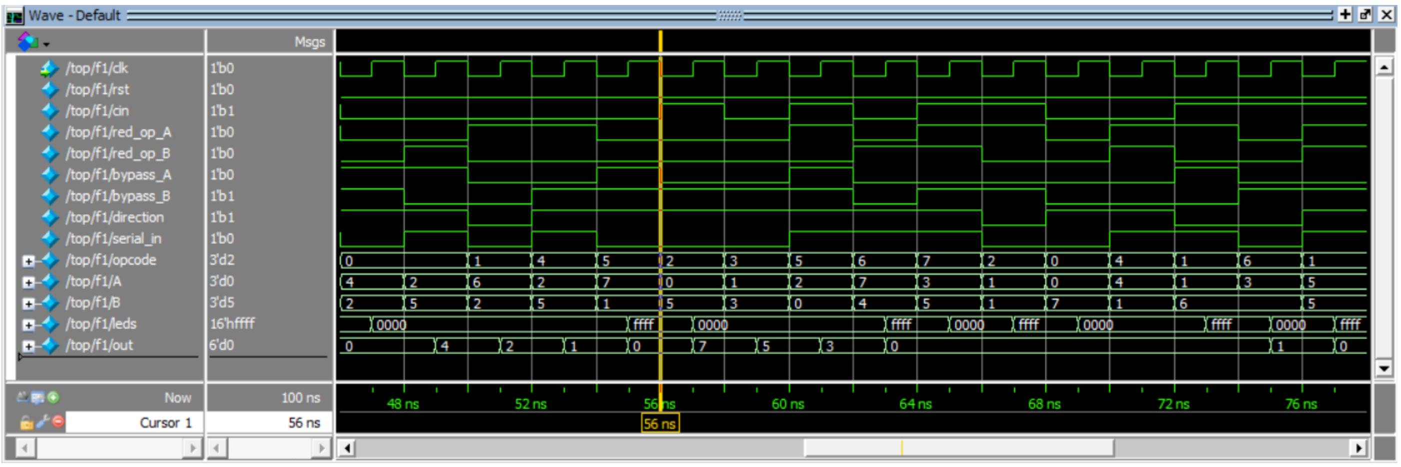
```
vlib work
vlog -f src_files_2.list
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all
add wave /top/f1/*
run -all
```



src_files_2.list

```
alsu_env_2.sv
alsu_if_2.sv
alsu_test_2.sv
ALSU.v
alsu_driver_2.sv
alsu_top_2.sv
```

Waveform



Transcript

```
# UVM-1.1d
# (C) 2007-2013 Mentor Graphics Corporation
# (C) 2007-2013 Cadence Design Systems, Inc.
# (C) 2006-2013 Synopsys, Inc.
# (C) 2011-2013 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(all)
# UVM_INFO @ 0: reporter [RNTST] Running test alsu_test...
# UVM_INFO alsu_test_2.sv(26) @ 100: uvm_test_top [run_phase] inside the ALSU test
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 100: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    5
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [run_phase]     1
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 100 ns  Iteration: 54  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

Question 3

Part #3:

In this part, you will use a configuration object to store the virtual interface value such that any component can retrieve the configuration object and read the values set in it.

Files to create:

1. top module
2. alsu_test
3. alsu_env
4. alsu_driver
5. alsu_config_obj

Steps:

1. Create top module where you will instantiate the DUT, interface, assign the interface to the DUT, generate the clock. Inside the initial block, set the virtual interface in the configuration database (uvm_config_db) using the set method then use the global task run_test to run your uvm test environment (alsu_test).
2. Create a class named alsu_config_obj that extends uvm_object
    a. Inside this UVM object, you will declare a virtual interface named **alsu_config_vif.**
3. Create a class named alsu_test in a separate file in a package, Inside the test class
    a. Declare a configuration object handle from alsu_config_obj named **alsu_config_obj_test** and UVM env handle from alsu_env
    b. In the build_phase, you will do the following
        i. Retrieve the virtual interface from the configuration database (uvm_config_db) and pass the value of the virtual interface to **alsu_config_obj_test**.alsu_config_vif
        ii. Set the configuration object in the configuration database to all the components under the test class so that any component can retrieve it.
        iii. Build the environment component (alsu_env).
    c. In the run phase, raise the objection, wait for #100 then display a message "Inside the ALSU test" using `uvm_info. Drop the objection after it. Don't forget to import the alsu_env package.
4. Create a class named alsu_env. Inside the build_phase, build the driver (Don't forget to import the driver package)
5. Create a class named alsu_driver extends uvm_driver
    a. Declare a virtual interface named **alsu_driver_vif** and a configuration object named **alsu_config_obj_driver**
    b. In the build_phase:
        i. Retrieve the configuration object set by the alsu_test from the configuration database (uvm_config_db) and pass the value of the configuration object to **alsu_config_obj_driver**
    c. In the connect phase:
        i. Assign **alsu_config_obj_driver.** alsu_config_vif to **alsu_driver_vif**
    d. In the run_phase:
        i. Reset the ALSU, then in a forever loop use $random to randomize the inputs of the ALSU using the virtual interface **alsu_driver_vif.**

Deliverables:

1. Simulate the top module and make sure that the message is displayed. Take a screenshot to use it in your PDF.
2. Add the signals of the interface to wave and make sure that the inputs are driven. Take a screenshot and add it to you PDF.

Top Module

```
import uvm_pkg: :*;
import alsu_test_pkg: :*;
`include "uvm_macros. svh"
module top ();
bit clk;
initial
begin
clk = 0;
forever
#1 clk = ~clk;
end

alsu_if f1(clk);
ALSU A(f1. A, f1. B, f1. cin, f1. serial_in, f1. red_op_A, f1. red_op_B, f1. opcode, f1. bypass_A
f1. bypass_B, f1. clk, f1. rst, f1. direction, f1. leds, f1. out);
initial
begin
uvm_config_db#(virtual alsu_if): : set(null, "uvm_test_top", "ALSU_IF", f1);

run_test("alsu_test");
end

endmodule
```

Interface

```
interface alsu_if (clk);
  input clk;
logic  rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
logic [2: 0] opcode;
logic signed [2: 0] A, B;
logic  [15: 0] leds;
logic  signed [5: 0] out;
endinterface
```

ALSU Test

```
 package alsu_test_pkg;
import uvm_pkg: :*;
 import alsu_env_pkg: :*;
 import alsu_config_pkg: :*;
 `include "uvm_macros. svh"
class alsu_test extends uvm_test;
   `uvm_component_utils(alsu_test)
   alsu_env env;
   virtual alsu_if alsu_test_vif;
   alsu_config_obj alsu_config_obj_test;

   function new(string name = "alsu_test", uvm_component parent  = null);
      super. new(name, parent);
   endfunction

   function void build_phase(uvm_phase phase);
      super. build_phase (phase);
      alsu_config_obj_test = alsu_config_obj: : type_id: : create("alsu_config_obj_test", this);
      env = alsu_env: : type_id: : create("env", this);
uvm_config_db#(virtual alsu_if): : get(this, "", "ALSU_IF", alsu_config_obj_test. alsu_config_vif);

      uvm_config_db#(alsu_config_obj): : set(this, " * ", "ALSU_IF_1", alsu_config_obj_test);
   endfunction

   task run_phase(uvm_phase phase);
      super. run_phase(phase);
      phase. raise_objection(this);
      #100; `uvm_info("run_phase", "inside the ALSU test", UVM_MEDIUM)
      phase. drop_objection(this);
   endtask
endclass

endpackage
```

ALSU Environment

```
package alsu_env_pkg;
   import alsu_driver_pkg::*;
   import uvm_pkg::*;

   `include "uvm_macros.svh"
   class alsu_env extends uvm_env;
      `uvm_component_utils(alsu_env)

alsu_driver driver;

      function new (string name = "alsu_env", uvm_component parent  = null);
         super.new(name, parent);
      endfunction

function void build_phase(uvm_phase phase);
   super.build_phase(phase);
driver = alsu_driver::type_id::create("driver", this);
   endfunction
endclass
endpackage
```

ALSU Driver

```
package alsu_driver_pkg;
   import uvm_pkg::*;
   import alsu_config_pkg::*;
   `include "uvm_macros.svh"
class alsu_driver extends uvm_driver;
   `uvm_component_utils(alsu_driver)
   virtual alsu_if alsu_driver_vif;
   alsu_config_obj alsu_config_obj_driver;
   function new(string name = "alsu_driver", uvm_component parent = null);
      super.new(name, parent);
   endfunction

   function void build_phase(uvm_phase phase);
      super.build_phase (phase);
      uvm_config_db #(alsu_config_obj)::get(this, "", "ALSU_IF_1", alsu_config_obj_driver);
   alsu_driver_vif = alsu_config_obj_driver.alsu_config_vif;
   endfunction
```

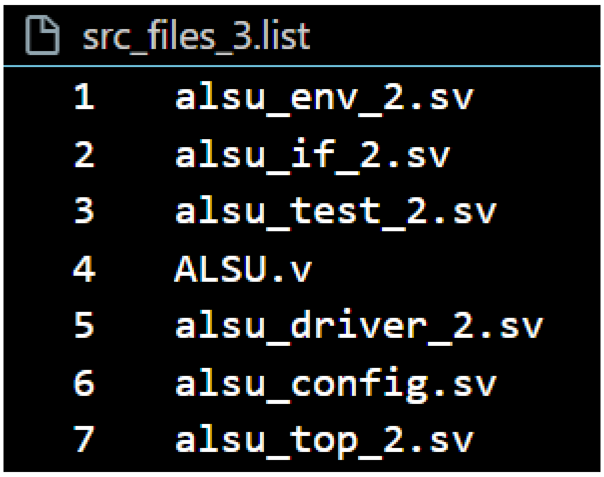
```
  task run_phase(uvm_phase phase);
    alsu_driver_vif.rst = 1;
    @(negedge alsu_driver_vif.clk);
    alsu_driver_vif.rst = 0;
    forever
    begin
    @(negedge alsu_driver_vif.clk);
    alsu_driver_vif.A  =  $random;
        alsu_driver_vif.B  =  $random;
        alsu_driver_vif.cin  =  $random;
        alsu_driver_vif.opcode  =  $random;
        alsu_driver_vif.bypass_A  =  $random;
        alsu_driver_vif.bypass_B  =  $random;
        alsu_driver_vif.red_op_A  =  $random;
        alsu_driver_vif.red_op_B  =  $random;
        alsu_driver_vif.direction  =  $random;
        alsu_driver_vif.serial_in  =  $random;
    end
   endtask
  endclass
endpackage
```

ALSU Configuration

```
package alsu_config_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
class alsu_config_obj extends uvm_object;
`uvm_object_utils(alsu_config_obj)
virtual alsu_if alsu_config_vif;

function new(string name = "alsu_config_obj");
super.new(name);

endfunction
endclass

endpackage
```

Do File

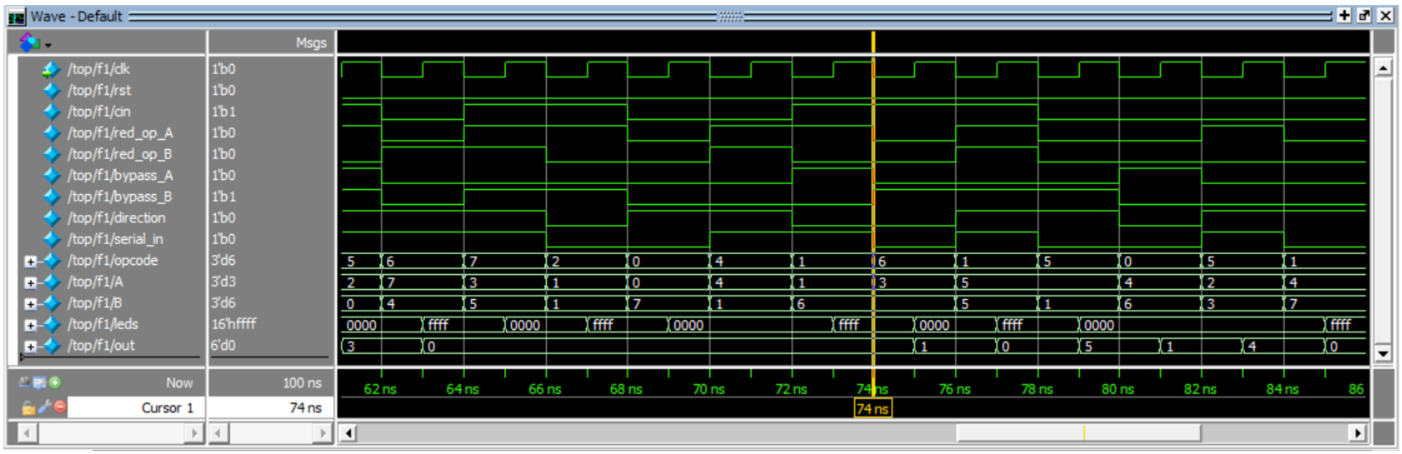
```
vlib work
vlog -f src_files_3.list
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all
add wave /top/f1/*
run -all
```



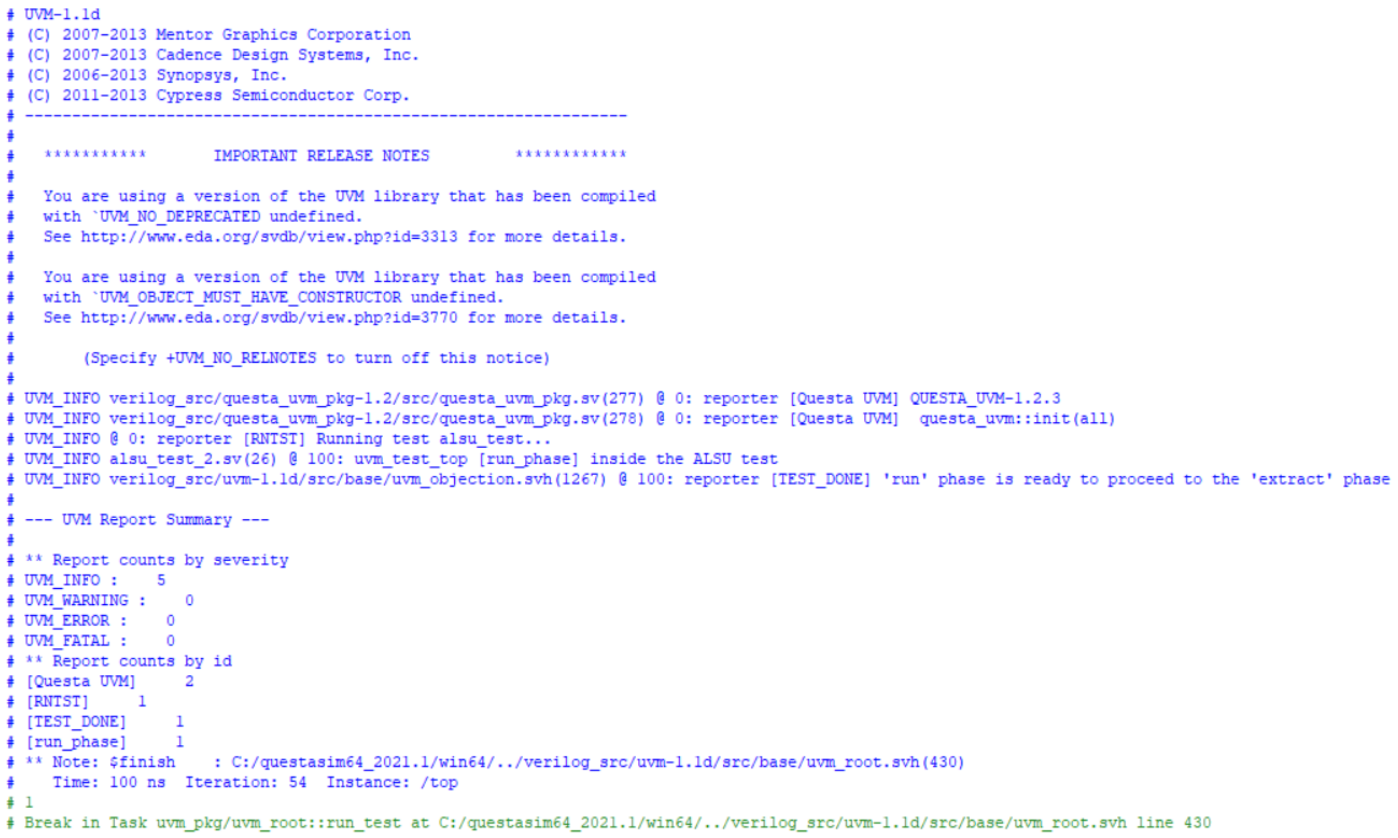
src_files_3.list

```
alsu_env_2.sv
alsu_if_2.sv
alsu_test_2.sv
ALSU.v
alsu_driver_2.sv
alsu_config.sv
alsu_top_2.sv
```

Waveform



Transcript



```
# UVM-1.1d
# (C) 2007-2013 Mentor Graphics Corporation
# (C) 2007-2013 Cadence Design Systems, Inc.
# (C) 2006-2013 Synopsys, Inc.
# (C) 2011-2013 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(all)
# UVM_INFO @ 0: reporter [RNTST] Running test alsu_test...
# UVM_INFO alsu_test_2.sv(26) @ 100: uvm_test_top [run_phase] inside the ALSU test
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 100: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    5
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [run_phase]     1
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 100 ns  Iteration: 54  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```