UVM

Quick Reference Guide

Author: Putta Satish

Email id: techsupport_vm@maven-silicon.com

www.maven-silicon.com

Maven Silicon Confidential

All the presentations, books, documents [hard copies and soft copies] labs and projects [source code] that you are using and developing as part of the training course are the proprietary work of Maven Silicon and it is fully protected under copyright and trade secret laws. You may not view, use, disclose, copy, or distribute the materials or any information except pursuant to a valid written license from Maven Silicon

Table of Contents

1. UVM TB Architecture 7
1.1 UVM TB Architecture 7
2. UVM Factory 8
2.1 Introduction 8
2.1.1 Steps to Use Factory 8
2.2 Factory Registration 8
2.2.1 `uvm_component_utils 8
2.2.2 `uvm_object_utils 8
2.2.3 `uvm_component_param_utils 9
2.2.4 `uvm_object_param_utils 9
2.3 Constructor Definitions for UVM components and objects 9
2.3.1 constructor of a component 9
2.3.2 constructor of an object 10
2.4 Factory Overriding 10
2.4.1 set_type_override_by_type 10
2.4.2 set_inst_override_by_type 10
2.5 Create Method 11
3. Stimulus Modelling 11
3.1 Introduction - Stimulus 11
3.1.1 Steps to define stimulus 11
3.2 Field Registration Macros 11
3.2.1 List of Field Registration Macros Available 11
3.2.2 Definition of a transaction class with Macros 12
3.3 Methods of Transaction class 12
3.3.1 copy & do_copy method 12
3.3.2 clone method 13
3.3.3 compare & do_compare method 13
3.3.4 print & do_print method 13

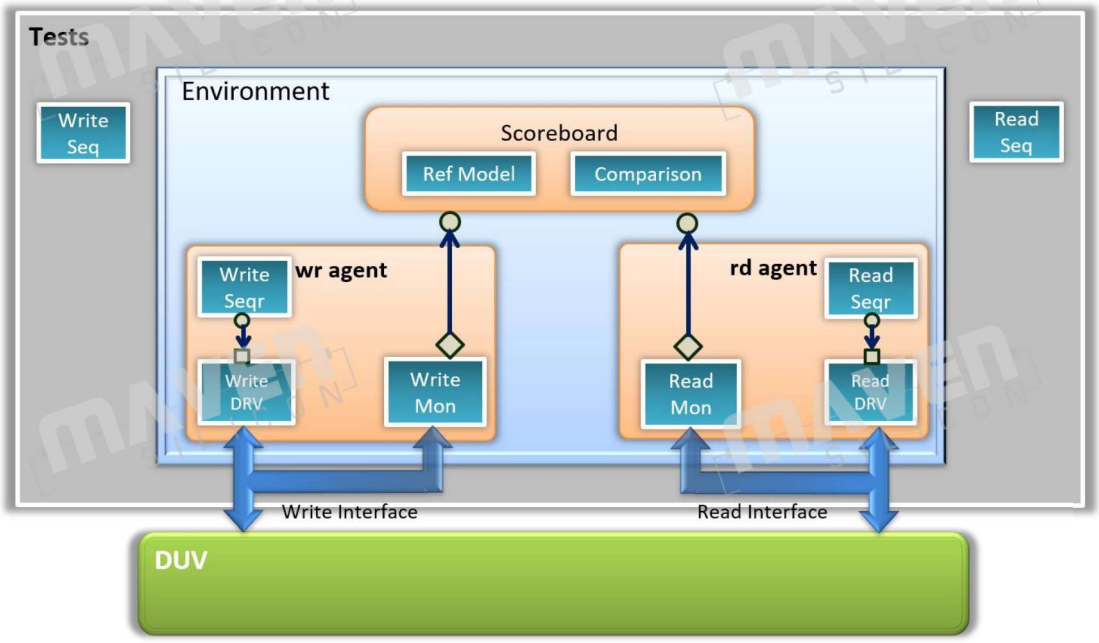
4. UVM Phases 14
4.1 Introduction 14
4.2 Pre run phases 14
4.2.1 build_phase 14
4.2.2 connect_phase 14
4.2.3 end_of_elaboration_phase 14
4.2.4 start_of_simulation_phase 14
4.3 run_phase 15
4.4 Post Run Phases 15
4.4.1 extract_phase 15
4.4.2 check_phase 15
4.4.3 report_phase 15
4.4.4 final_phase 15
4.5 Objection Mechanism 15
5. Reporting Mechanism 16
5.1 Introduction 16
5.2 UVM Reporting Macros 16
5.3 Verbosity Settings 16
5.4 Command line options 17
5.4.1 +UVM_VERBOSITY 17
5.4.2 +UVM_REPORT_DISABLE_FILE_LINE 17
6. TLM – Transaction Level Modelling 18
6.1 Introduction 18
6.2 Blocking Get Port 18
6.3 Blocking Put Port 19
6.4 Connecting TLM Ports 20
6.5 Analysis Ports 20
6.6 Analysis FIFO 21
7. UVM Configuration 22
7.1 Introduction 22
7.2 Example 22
7.2.1 Configuration Object 22
7.2.2 Setting Configuration Object 23
7.2.3 Getting Configuration Object 23

8. UVM Sequences 23
8.1 Introduction 23
8.1.1 Steps to Define Sequence 23
8.2 Sequence Item flow from sequence to driver 24
8.3 Sample Sequence Definition 24
8.4 Sample Sequence Definition with do macros 25
8.5 Sample Sequence Definition with inline constraints 25
8.6 Sequence – Configuration 26
8.7 Setting Default Sequence 26
8.8 Sequence Objection 26
8.9 Sequence Library Definition 27
8.10 Testcase with Sequence Library 27
9. Virtual Sequences & Virtual Sequencers 28
9.1 Introduction 28
9.2 Stand-alone Virtual Sequence 28
9.3 Testcase with Stand-alone Virtual Sequence 28
9.4 Sample Virtual Sequencer 29
9.5 Base Virtual Sequence 29
9.6 Extended Virtual Sequence 30
9.7 Sub sequencer Connection 30
9.8 Testcase with Virtual Sequence 31
10. UVM Callbacks 32
10.1 Introduction 32
10.2 Example of a transactor without callbacks 32
10.2.1 Step 1: Definition of driver 32
10.2.2 Step 2: Definition of test to run simulation 32
10.2.3 Step 3: Expected Output 32
10.3 Example of a transactor with callbacks 33
10.3.1 Step 1: Definition of driver callback class with empty callback methods 33
10.3.2 Step 2: Definition of driver class with empty callback hooks 33
10.3.3 Step 3: Definition of extended driver callback class with callback methods 34
10.3.4 Step 4: Linking Driver and Callback classes 34
10.3.5 Expected output 34

11. UVM Events 35
11.1 Introduction 35
11.2 Example to demonstrate the usage of uvm events 35
11.2.1 Step 1: Configuration Class with event pool 35
11.2.2 Step 2: Definition of event callback class 35
11.2.3 Step 3: Definition of Driver with UVM events 35
11.2.4 Step 4: Definition of Monitor with UVM events 36
11.2.5 Expected Output 36
12. Verification of Modulo – 13 Loadable UP counter 37
12.1 Block Diagram 37
12.2 TB Architecture 37
12.3 Interface 37
12.4 Transaction Class (Sequence Item) 38
12.5 Configuration Class 38
12.6 Components in the input agent side 39
12.6.1 Input Driver 39
12.6.2 Input Sequencer 39
12.6.3 Input Monitor 40
12.6.4 Input Agent 41
12.7 Components in the output agent side 42
12.7.1 Output Monitor 42
12.7.2 Output Agent 42
12.8 Scoreboard with Reference Model Logic 43
12.9 Environment 45
12.10 Sequence 45
12.11 Base testcase 46
12.12 Derived testcase 46
12.13 Package 47
12.14 Top Module 47

1. UVM TB Architecture

1.1 UVM TB Architecture



2. UVM Factory

2.1 Introduction

Factory is a class that manufactures (creates) UVM components and objects during runtime and it has the ability to modify the types and number of objects that create the testbench hierarchy.

2.1.1 Steps to Use Factory

- ✓ Register class types with the factory
- ✓ Override Components and Objects if needed
- ✓ Create components and objects using factory

2.2 Factory Registration

Every class (component or object) must be registered by using **`uvm_component_utils** or **`uvm_object_utils** or **`uvm_component_param_utils** or **`uvm_object_param_utils** macros.

2.2.1 `uvm_component_utils

Macro for registering a component.

Syntax :

```
`uvm_component_utils (component_class_name)
```

Example :

```
class bus_wr_driver extends uvm_driver #(write_xtn);
    `uvm_component_utils(bus_wr_driver)

    // Driver definition
endclass : bus_wr_driver
```

2.2.2 `uvm_object_utils

Macro for registering an object.

Syntax :

```
`uvm_object_utils (object_class_name)
```

Example :

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

// write_xtn definition
endclass : write_xtn
```

2.2.3 `uvm_component_param_utils

Macro for registering a parameterized component.

Syntax :

```
`uvm_component_param_utils (component_class_name
                            #(list of parameters));
```

Example :

```
class bus_wr_driver #(Type T=int,width W=32)
                     extends uvm_driver #(write_xtn);
   `uvm_component_param_utils(bus_wr_driver #(T,W))

    // Driver definition
endclass : bus_wr_driver
```

2.2.4 `uvm_object_param_utils

Macro for registering a parameterized object.

Syntax :

```
`uvm_object_param_utils (object_class_name
                         #(list of parameters))
```

Example :

```
class write_xtn #(type T=int)
                extends uvm_sequence_item;
  `uvm_object_param_utils(write_xtn #(T))

    // write_xtn definition
endclass : write_xtn
```

2.3 Constructor Definitions for UVM components and objects

The uvm_component and uvm_object constructors (function new()) are virtual methods that should be mandatorily defined with proper arguments

2.3.1 constructor of a component

Example :

```
class my_component extends uvm_component;
   function new(string name="my_component",
                uvm_component parent=null);
      super.new(name,parent);
   endfunction : new
endclass : my_component
```

2.3.2 constructor of an object

Example :

```
class my_config_obj extends uvm_object;
   function new(string name="my_config_obj");
      super.new(name);
   endfunction : new
endclass : my_config_obj
```

2.4 Factory Overriding

Used to substitute a class with another class of a derived type when it is constructed

2.4.1 set_type_override_by_type

Syntax :

```
factory.set_type_override_by_type(original_class_name::get_type(),
                                  substitue_class_name::get_type(),
                                  replace = 1);
```

Example :

```
class test extends uvm_test;
    virtual function build_phase(uvm_phase phase);
        factory.set_type_override_by_type ( bus_wr_driver::get_type(),
                                            apb_wr_driver::get_type());
    endfunction : build_phase
endclass : test
```

2.4.2 set_inst_override_by_type

Syntax :

```
set_inst_override_by_type("instance_path",
                          original_class_name::get_type(),
                          substitue_class_name::get_type(),
                          replace = 1);
```

Example :

```
class test extends uvm_test;
   virtual function build_phase(uvm_phase phase);
      set_inst_override_by_type("*.agent1.*",
                                bus_wr_driver::get_type(),
                                apb_wr_driver::get_type());
   endfunction : build_phase
endclass : test
```

2.5 Create Method

Components and objects are created using UVM's factory method 'create' instead of the default constructor function 'new' so that, factory overriding is possible.

Syntax :

```
//for components
obj_name = class_name::type_id::create("obj_name", this);

//for objects like sequences and sequence items
obj_name = class_name::type_id::create("obj_name");
```

Example :

```
class bus_wr_agent extends uvm_agent;
    bus_wr_driver drvh;

    function build_phase(uvm_phase phase);
        super.build_phase(uvm_phase phase);
        drvh=bus_wr_driver::type_id::create("drvh",this);
    endfunction : build_phase
endclass : bus_wr_agent
```

3. Stimulus Modelling

3.1 Introduction - Stimulus

It represents the main transaction input to the DUT based on the protocol of the Design.

3.1.1 Steps to define stimulus

- ✓ Derive a data item class from uvm_sequence_item base class
- ✓ Register with the factory using `uvm_object_utils macro
- ✓ Define a constructor for the data item
- ✓ Add control fields for the items such as constraints
- ✓ UVM field macros to enable printing, copying, comparing, etc

3.2 Field Registration Macros

They are used to register the class properties (fields) so that, these properties will be enabled for various pre-defined methods like copy, compare and print etc.

3.2.1 List of Field Registration Macros Available

`uvm_field_int	`uvm_field_array_int
`uvm_field_object	`uvm_field_array_object
`uvm_field_string	`uvm_field_array_string
`uvm_field_enum	`uvm_field_queue_int
`uvm_field_real	`uvm_field_queue_object
`uvm_field_event	`uvm_field_queue_string

3.2.2 Definition of a transaction class with Macros

Example :

```
class write_xtn extends uvm_sequence_item;

    rand bit[`RAM_WIDTH-1 : 0] data;
    rand bit[`ADDR_SIZE-1 : 0] address;
    rand bit write;
    rand addr_t xtn_type;
    rand bit[63:0] xtn_delay;

    `uvm_object_utils_begin(write_xtn)
          `uvm_field_int(data,UVM_ALL_ON)
          `uvm_field_int(address,UVM_ALL_ON)
          `uvm_field_int(write,UVM_ALL_ON)
          `uvm_field_enum(addr_t,xtn_type,UVM_ALL_ON)
          `uvm_field_int(xtn_delay,UVM_ALL_ON+UVM_NOCOPY)
    `uvm_object_utils_end

    function new(string name = "write_xtn");
       super.new(name);
    endfunction : new

    constraint valid_wr{address inside {[0:4095]};}

endclass : write_xtn
```

3.3 Methods of Transaction class

3.3.1 copy & do_copy method

Example :

```
// Defnition of do_copy method
virtual function void do_copy(uvm_object rhs);
    write_xtn rhs_;
    if(!$cast(rhs_,rhs))
        begin
            `uvm_fatal("do_copy","cast of rhs object failed")
        end
    super.do_copy(rhs);
    this.data=rhs_.data;
    this.address=rhs_.address;
    this.write=rhs_.write;
endfunction : do_copy

// Usage of copy method
write_xtn wr_xtnh;
write_xtn wr_copy_xtnh;

wr_xtnh = write_xtn::type_id::create("wr_xtnh");
wr_copy_xtnh = write_xtn::type_id::create("wr_copy_xtnh");

assert(wr_xtnh.randomize());
wr_copy_xtnh.copy(wr_xtnh);
```

3.3.2 clone method

Example :

```
// Usage of do_compare method
write_xtn write_xtnh2;
write_xtn write_xtnh1 = write_xtn::type_id::create("write_xtnh1");
assert(write_xtnh1.randomize());
$cast(write_xtnh2,write_xtnh1.clone());
```

3.3.3 compare & do_compare method

Example :

```
// Defnition of do_compare method
virtual function bit do_compare(uvm_object rhs,uvm_comparer comparer);
   write_xtn rhs_;
   if(!$cast(rhs_,rhs))
        begin
            `uvm_error("do_compare","cast of rhs object failed")
            return 0;
        end
   return
        super.do_compare(rhs,comparer) &&
        this.data == rhs_.data &&
        this.address == rhs_.address &&
        this.write == rhs_.write;
endfunction : do_compare


// Usage of do_compare method
write_xtn w_xtnh1=write_xtn::type_id::create("w_xtnh1");
write_xtn w_xtnh2=write_xtn::type_id::create("w_xtnh2");

//compare w_xtnh1 and w_xtn2
if(!w_xtnh1.compare(w_xtnh2))
    $display("Mismtach between h1 and h2");
```

3.3.4 print & do_print method

Example :

```
// Usage of do_print method
virtual function void do_print(uvm_printer  printer);
   printer.print_field("new_data",new_data, 32,  UVM_DEC);
   printer.print_object("new_handle", new_handle);
endfunction : do_print

// Usage of do_print method
write_xtn wr_packet;
//construct wr_packet
wr_packet = write_xtn::type_id::create("wr_packet");
//randomize wr_packet and print
assert(wr_packet.randomize());
wr_packet.print(); //Prints in table view
wr_packet.print(uvm_default_tree_printer);
```

4. UVM Phases

4.1 Introduction

Phases help in synchronizing different events between Multiple Agents. The "run_test" is the method to initiate execution of all the phases.

4.2 Pre run phases

4.2.1 build_phase

Example :

```
class ram_write_agent extends uvm_agent;

    ram_write_monitor monh;

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        monh=ram_write_monitor::type_id::create("monh",this);
    endfunction : build_phase

endclass : ram_write_agent
```

4.2.2 connect_phase

Example :

```
class ram_write_agent extends uvm_agent;

    uvm_active_passive_enum is_active;
    ram_write_sequencer seqrh;
    ram_write_driver drvh;

    virtual function void connect_phase(uvm_phase phase);
        if(is_active == UVM_ACTIVE)
            drvh.seq_item_port.connect(seqrh.seq_item_export);
    endfunction : connect_phase

endclass : ram_write_agent
```

4.2.3 end_of_elaboration_phase

Example :

```
virtual function void end_of_elaboration_phase(uvm_phase phase);
    `uvm_info(get_type_name(),"start of simulation",UVM_LOW)
endfunction : end_of_elaboration_phase
```

4.2.4 start_of_simulation_phase

Example :

```
virtual function void start_of_simulation_phase(uvm_phase phase);
   uvm_top.print_topology();
endfunction : start_of_simulation_phase
```

4.3 run_phase

Example :

```
class apb_mon extends uvm_monitor;
   task run_phase(uvm_phase phase);
        write_xtn xtn;
       forever
           begin
               //collect_data
           end
    endtask : run_phase
endclass : apb_mon
```

4.4 Post Run Phases

4.4.1 extract_phase

retrieves and processes information from scoreboards and functional coverage monitors and it will be executed in Bottom up manner

4.4.2 check_phase

checks if the DUT behaved correctly and identifies the errors that may have occurred during the execution and it will be executed in Bottom up manner

4.4.3 report_phase

displays the result of the simulation and it will be executed in Bottom up manner

4.4.4 final_phase

completes any other outstanding actions and it will be executed in top-down manner.

4.5 Objection Mechanism

Example :

```
class agent extends uvm_agent;
   virtual task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      #100;
      phase.drop_objection(this);
   endtask : run_phase
endclass : agent
```

```
class driver extends uvm_driver;
   virtual task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      #10;
      phase.drop_objection(this);
   endtask : run_phase
endclass : driver
```

5. Reporting Mechanism

5.1 Introduction

Helps in debugging an complex verification environment using different severities like info, warning, error and fatal. Useful for displaying simulation results.

5.2 UVM Reporting Macros

Syntax:

```
`uvm_info(string id,string messsage,int verbosity=UVM_MEDIUM);
```

Example :

```
`uvm_info("INFO",$sformatf("val:%0d",val),UVM_HIGH)
`uvm_warning("WARNING","This is a warning")
`uvm_error("ERROR","This is a error")
`uvm_fatal("FATAL","A fatal error has occured")
```

5.3 Verbosity Settings

By changing the verbosity, printing the messages can be controlled. Below are the different verbosity levels in an order, least to highest. If there are multiple info statements of different verbosity levels, then, messages with a verbosity greater than the default verbosity are ignored

UVM_NONE
UVM_LOW
UVM_MEDIUM
UVM_HIGH
UVM_FULL

Example :

```
 `uvm_info(drv, "verbosity medium",UVM_MEDIUM);
 `uvm_info(drv, "verbosity warning",UVM_LOW);
 `uvm_info(drv, "verbosity error",UVM_NONE);

write_xtn xtn;
initial
    begin
        xtn=new("xtn");
        xtn.set_report_verbosity_level(UVM_LOW);

        //printing logic
    end

//Below is the expected result: // Verbority medium will not be displayed

// UVM_INFO ../wr_agt_top/ram_wr_driver.sv(50)@0:wr_agt_agth.drvh[drv] verbosity warning
// UVM_INFO ../wr_agt_top/ram_wr_driver.sv(51)@0:wr_agt_agth.drvh[drv] verbosity error
```

5.4 Command line options

5.4.1 +UVM_VERBOSITY

To set the required level of verbosity as the default verbosity during the simulation.

Example :

```
vsim work.top +UVM_VERBOSITY=UVM_MEDIUM
```

5.4.2 +UVM_REPORT_DISABLE_FILE_LINE

Disables file names and line number while reporting

Example :

```
vlog -work ../tb/top.sv +define+UVM_REPORT_DISABLE_FILE_LINE
```

6. TLM – Transaction Level Modelling

6.1 Introduction

TLM promotes reuse as they have same interface and supports interoperability for the mixed-language verification environments. It also Maximizes reuse and minimizes the time and effort.

6.2 Blocking Get Port

With blocking get port, the initiator gets the transactions through blocking get implementation port, from the target component.

Example :


```
class ahb_driver extends uvm_component;
    uvm_blocking_get_port #(write_xtn) get_port;
    function new(string name,uvm_component parent);
        super.new(name,parent);
        get_port=new("get_port",this);
    endfunction : new

    virtual task run_phase(uvm_phase phase);
        write_xtn xtn;
        for(int i=0;i<N;i++)
            begin
                get_port.get(xtn);
            end
    endtask :  run_phase
endclass : ahb_driver


class ahb_generator extends uvm_component;
    uvm_blocking_get_imp#(write_xtn, ahb_generator)get_imp;

    virtual task get(output write_xtn xtn_h);
        write_xtn xtn=new();
        //Randomize the xtn object
        xtn_h = xtn;
    endtask : get
endclass : ahb_gen
```

6.3 Blocking Put Port

With blocking put port, the initiator sends the transactions through a blocking put implementation port, to the target component.

Example :


```
class apb_generator extends uvm_component;
    uvm_blocking_put_port #(write_xtn) put_port;

    function new(string name, uvm_component parent);
        put_port = new("put_port",this);
    endfunction : new

    virtual task run_phase(uvm_phase phase);
        write_xtn xtn;
        for(int i=0;i<N;i++)
            begin
                // Create the object of xtn &
                // randomize
                put_port.put(xtn);
            end
    endtask : run_phase
endclass : ahb_gen

class apb_driver extends uvm_component;
    uvm_blocking_put_imp#(write_xtn, apb_driver) put_imp;

    virtual task put(write_xtn xtn);
        case(xtn.kind)
            READ :  // Do read
            WRITE:  // Do write
        endcase
    endtask : put
endclass : apb_driver
```

6.4 Connecting TLM Ports

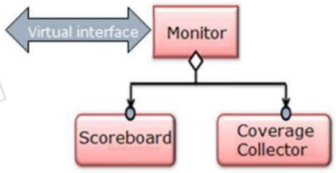
```
class my_env extends uvm_env;

    //Get
    ahb_generator ahb_genh;
    ahb_driver ahb_drvh;
    // Put
    apb_generator apb_genh;
    apb_driver apb_drvh;

    function void connect_phase(connect_phase phase);
        ahb_drvh.get_port.connect(ahb_genh.get_imp);
        apb_genh.put_port.connect(apb_drvh.put_imp);
    endfunction : connect_phase

endclass : my_env
```

6.5 Analysis Ports

Single Analysis Port and Multiple Analysis Exports

Analysis port may be connected to zero, one or many Analysis Exports.

Example :



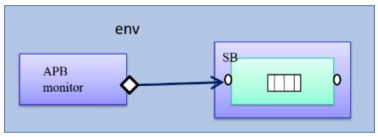

```
class apb_mon extends uvm_monitor;
    uvm_analysis_port #(write_xtn) ap;
    function new(string name,uvm_component  parent);
        super.new(name,parent);
        ap=new("ap",this);
    endfunction:new

    task run_phase(uvm_phase phase);
       write_xtn xtn;
       forever
          begin
             //sample interface signals
             xtn.addr=vif.addr;
             ap.write(xtn);
          end
    endtask:run_phase
endclass : apb_mon
```

```
class sb extends uvm_scoreboard;
    uvm_analysis_imp #(write_xtn,sb) analysis_imph;
    function void write(write_xtn xtn);
       // Record coverage information of xtn
    endfunction : write
endclass : sb

class my_env extends uvm_env;

    apb_mon apb_monh;
    sb sbh;
    coverage_collector covh;
    function void connect_phase(uvm_phase phase);
         apb_monh.ap.connect(sbh.analysis_imph);
         apb_monh.ap.connect(covh.analysis_imph);
    endfunction : connect_phase

endclass  : my_env
```

6.6 Analysis FIFO

Helps in achieving synchronization between the TB components.

Example :




```
class sb extends uvm_scoreboard;
    uvm_tlm_analysis_fifo#(write_xtn) fifo_h;

    function new( string name, uvm_component parent);
        fifo_h = new("fifo_h",this);
    endfunction : new

    virtual task run_phase(uvm_phase phase);
        write_xtn xtn;
        forever
            begin
                fifo_h.get(xtn);
                //process xtn
            end
    endtask : run_phase
endclass : sb
```

```
class apb_mon extends uvm_monitor;
    uvm_analysis_port#(write_xtn) ap;

    task run_phase(uvm_phase phase);
        write_xtn xtn;
        forever
            begin
                //write transaction
                ap.write(xtn);
            end
    endtask : run_phase
endclass : apb_mon


class apb_env extends uvm_env;
    apb_mon apb_monh;
    sb sbh;

    function connect_phase(uvm_phase phase);
        apb_monh.ap.connect(sbh.fifo_h.analysis_export);
    endfunction : connect_phase
endclass : apb_env
```

7. UVM Configuration

7.1 Introduction

Helps us to build a reusable testbench. Top Level component has the ability to configure the structure of lower level components.

7.2 Example

In the below example, agent is being configured to be an active type.

7.2.1 Configuration Object

```
class ram_config extends uvm_object;
    `uvm_object_utils(ram_config)
    //configuration parameters
    virtual ram_if vif;
    uvm_active_passive_enum is_active = UVM_ACTIVE;
    bit has_functional_coverage = 0;
    bit has_scoreboard  = 1;
    bit has_write_agent  = 1;
    bit has_read_agent  = 1;
    static int mon_rcvd_xtn_cnt = 0;
    int ram_verbosity;

    function new(string name = "ram_config");
        super.new(name);
    endfunction : new

endclass : ram_config
```

7.2.2 Setting Configuration Object

```
class ram_test extends uvm_test;
    `uvm_component_utils(ram_test)
    ram_write_agent ram_envh;
    ram_config tb_cfg;

    function void build_phase(uvm_phase phase);
        tb_cfg=ram_config::type_id::create("tb_cfg");
        tb_cfg.is_active = UVM_ACTIVE;
        tb_cfg.has_functional_coverage = 0;
        uvm_config_db #(ram_config)::set(this,"*","ram_config",tb_cfg)
        super.build_phase(phase);
        ram_envh=ram_write_agent::typ_id::create("ram_envh",this);
    endfunction : build_phase

endclass : ram_test
```

7.2.3 Getting Configuration Object

```
class ram_write_agent extends uvm_agent;

    ram_wr_driver driver;
    ram_wr_monitor monitor;
    ram_wr_sequencer sequencer;
    ram_config tb_cfg;

    function void build_phase(uvm_phase phase);
        if(!uvm_config_db #(ram_config)::get(this,"","ram_config",tb_cfg))
            `uvm_fatal("TB CONFIG","cannot get the tb_cfg from uvm_config")
        super.build_phase(phase);
        monitor=ram_wr_monitor::type_id::create("monitor",this);
        if(tb_cfg.is_active==UVM_ACTIVE)
            begin
                sequencer = ram_wr_sequencer::type_id::create("sequencer",this);
                driver =ram_wr_driver::type_id::create("driver",this);
            end
    endfunction : build_phase
endclass : ram_write_agent
```

8. UVM Sequences

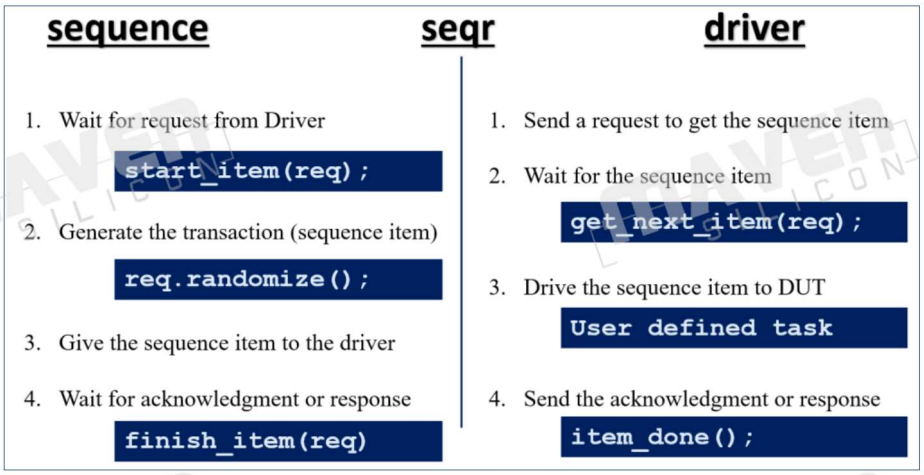
8.1 Introduction

It generates the stimulus based on constraints and sends the stimulus to driver through a sequencer. They can be nested and be overridden with factory.

8.1.1 Steps to Define Sequence

- ✓ Extend from uvm_sequence
- ✓ Define pre_body, body and post_body tasks.

8.2 Sequence Item flow from sequence to driver




```
//Sequence Definition
task body();
    repeat(10)
        begin
            start_item(req);
            assert(req.randomize());
            finish_item(req);
        end
endtask : body
```

```
//Driver Definition
task run_phase(uvm_phase);
   forever
      begin
          seq_item_port.get_next_item(req);
         //User defined task
          seq_item_port.item_done();
      end
endtask : run_phase
```

```
//Testcase Definition
task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    seq_h.start(env_h.agt_h.seqr_h);
    phase.drop_objection(this);
endtask : run_phase
```

8.3 Sample Sequence Definition

```
class ram_do_seq extends uvm_sequence#(write_xtn);
  `uvm_object_utils(ram_do_seq)
   function new(string name="ram_do_seq");
     super.new(name);
   endfunction : new

   virtual task body();
     repeat(10)
       begin
         req=write_xtn::type_id::create("req");
         start_item(req);
         assert(req.randomize());
         finish_item(req);
       end
   endtask : body
endclass : ram_do_seq
```

8.4 Sample Sequence Definition with do macros

```
class ram_sub_seq extends uvm_sequence#(write_xtn);
   //Required macro for automation
  `uvm_object_utils(ram_sub_seq)
   ram_do_seq seq_do;

   function new(string name="ram_sub_seq");
      super.new(name);
   endfunction:new

   virtual task body();
      repeat(10)
         begin
            `uvm_do(req);
            `uvm_do(seq_do);
         end
   endtask:body
endclass:ram_sub_seq
```

8.5 Sample Sequence Definition with inline constraints

```
class ram_do_with_seq extends uvm_sequence#(write_xtn);
   //Required macro for automation
  `uvm_object_utils(ram_do_with_seq)

   function new(string name="ram_do_with_seq");
      super.new(name);
   endfunction:new

   virtual task body();
      repeat(10)
         begin
            //`uvm_do_with(req,{address==0;data==32'hff;})
            req=write_xtn::type_id::create("req");
            start_item(req);
            assert(req.randomize() with {address==0;data==32'hff;})
            finish_item(req);
         end
   endtask:body
endclass:ram_do_with_seq
```

8.6 Sequence – Configuration

```
class ram_seq extends uvm_sequence#(write_xtn);

   function new(string name="ram_seq");
      super.new(name);
   endfunction:new
   //Automation
   `uvm_object_utils(ram_seq)
   ram_config m_cfg;

   task body();
      if(!uvm_config_db#(ram_config)::get(null, get_full_name(),
                                          "ram_config", m_cfg))
        `uvm_fatal("RAM_SEQ","cannot get the config m_cfg from the database")
      req=write_xtn::type_id::create("req");
      repeat(m_cfg.count)
        begin
          start_item(req);
          assert(req.randomize());
          finish_item(req);
        end
   endtask:body
endclass:ram_seq
```

8.7 Setting Default Sequence

```
//Using Wrapper
uvm_config_db#(uvm_object_wrapper)::set(this,"agent.seqr.run_phase",
               "default_sequence",
               seq1::type_id::get());


//Using Instance
seq1_h = seq1::type_id::create("seq1_h");
uvm_config_db#(uvm_sequence_base)::set(this,"agent.seqr.main_phase",
                                       "default_sequence",seq1_h);
```

8.8 Sequence Objection

```
class seq1 extends uvm_sequence#(write_xtn);

  virtual task body();
    if(starting_phase!=null)
       starting_phase.raise_objection(this,"starting my_seq");
       //body of sequence
    if(starting_phase!=null)
       starting_phase.drop_objection(this,"Ending my_seq");
  endtask:body

endclass:seq1
```

8.9 Sequence Library Definition

```
//Sequence Library Definition
class my_seq_lib extends uvm_sequence_library #(write_xtn);
   `uvm_object_utils(my_seq_lib)
   `uvm_sequence_library_utils(my_seq_lib)
    function new(string name="")
      super.new(name);
       init_sequence_libray();
    endfunction:new
endclass:my_seq_lib

class seq1 extends my_seq;
   `uvm_object_utils(seq1)
   `uvm_add_to_seq_lib(seq1,my_seq_lib)
endclass:seq1

class seq2 extends my_seq;
   `uvm_object_utils(seq2)
   `uvm_add_to_seq_lib(seq2,my_seq_lib)
endclass:seq2
```

8.10 Testcase with Sequence Library

```
//Testcase with Sequence Library
class test_seq_lib extends base_test;

    task run_phase();

       phase.raise_objection(this);

       my_seq_lib seq_lib = my_seq_lib::type_id::create("seq_lib");
       seq_lib.min_random_count = 5;
       seq_lib.max_random_count = 15;
       seq_lib.selection_mode = UVM_SEQ_LIB_RANDC;
       // register a new sequence to the library
       seq_lib.add_sequence(err_seq::get_type());
       assert(seq_lib.randomize());
       seq_lib.start(envh.agnth.seqrh);

       phase.drop_objection(this);

    endtask : run_phase
endclass : test_seq_lib
```

9. Virtual Sequences & Virtual Sequencers

9.1 Introduction

They improve the reusability of sequences and testbenches. Virtual Sequence can be used as a stand-alone object and used along with the virtual sequencers.

9.2 Stand-alone Virtual Sequence

```
class virtual_sequence extends uvm_sequence#(uvm_sequence_item);

    `uvm_object_utils(virtual_sequence)

    ram_wr_sequencer wr_seqrh;
    ram_rd_sequencer rd_seqrh;
    write_odd_seq w_odd_seqh;
    read_odd_seq r_odd_seqh;

    function new(string name="virtual sequence");
        super.new(name);
    endfunction : new

    task body();
        w_odd_seqh=write_odd_seq::type_id::create("w_odd_seqh");
        r_odd_seqh=read_odd_seqh::type_id::create("r_odd_seqh");
        //Run the sub-sequence
        w_odd_seqh.start(wr_seqrh);
        r_odd_seqh.start(rd_seqrh);
        w_odd_seqh.start(wr_seqrh);
    endtask : body

endclass : virtual_sequence
```

9.3 Testcase with Stand-alone Virtual Sequence

```
class test_vseq extends uvm_test;

    //UVM automation and constructor
    virtual_sequence v_seqh;

    task run_phase(uvm_phase phase);

        phase.raise_objection(this);
        v_seqh=virtual_sequence::type_id::create("v_seqh");
        v_seqh.wr_seqrh=env.wagent.write_agent.seqrh;
        v_seqh.rd_seqrh=env.ragent.read_agent.seqrh;
        v_seqh.start(null);
        phase.drop_objection(this);

    endtask : run_phase

endclass:test_vseq
```

9.4 Sample Virtual Sequencer

```
class ram_virtual_sequencer extends uvm_sequencer#(uvm_sequence_item);

    ram_wr_sequencer wr_seqrh;
    ram_rd_sequencer rd_seqrh;

    //constructor

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

    //UVM automation macros for sequencers

    `uvm_component_utils(ram_virtual_sequencer)

endclass : ram_virtual_sequencer
```

9.5 Base Virtual Sequence

```
class virtual_sequence_base extends uvm_sequence #(uvm_sequence_item);
    `uvm_object_utils(virtual_sequence_base)
     //This is needed to get the sub_sequencer in the m_sequence
     virtual_sequencer v_sqrh;
     //Local sub-sequencer handles
     ram_wr_sequencer w_sqrh;
     ram_rd_sequencer r_sqrh;
     function new(string name="virtual_sequence_base");
         super.new(name);
     endfunction : new

     //Assign pointers to the sub-sequencer in the base body method
     task body();
        if(!$cast(v_sqrh,m_sequencer))
            `uvm_error(get_full_name(),"virtual sequencer pointer
        w_sqrh=v_sqrh.wr_seqrh;
        r_sqrh=v_sqrh.rd_seqrh;
     endtask : body

endclass : virtual_sequence_base
```

9.6 Extended Virtual Sequence

```
class random_virtual_seq extends virtual_sequence_base;

     random_write_seq rand_wseqh;
     random_read_seq rand_rseqh;

     task body;
          super.body;
             rand_wseqh=random_write_seq::type_id::create("rand_wseqh");
             rand_rseqh=random_read_seq::type_id::create("rand_rseqh");
          repeat(20)
              begin
                  rand_wseqh.start(w_sqrh);
                  rand_rseqh.start(r_sqrh);
              end
     endtask : body

 endclass : random_virtual_seq
```

9.7 Sub sequencer Connection

```
class ram_tb extends uvm_env;
     ram_write_agent wagenth;
     ram_read_agent ragenth;
     ram_virtual_sequencer v_seqrh;
     //construnctor and uvm automation macros
     virtual function void build_phase(uvm_phase phase);
          super.build_phase(phase);
          //Build env with subsequencers
          wagenth=ram_write_agent::type_id::create("wagenth",this);
          ragenth=ram_read_agent::type_id::create("ragenth",this);
          //Build the virtual sequencer
          v_seqrh=ram_virtual_sequencer::type_id::create("v_seqrh",this);

     endfunction : build_phase

      //connect virtual sequencer to subsequencers.
     function void connect_phase(uvm_phase phase);
          v_seqrh.wr_seqrh=wagenth.write_agent.seqrh;
          v_seqrh.rd_seqrh=ragent.read_agent.seqrh;
     endfunction:connect_phase
endclass : ram_tb
```

9.8 Testcase with Virtual Sequence

```
class test_ram_virt_seq extends base_test;

    //UVM automation and constructor
    random_virtual_seq test_seqh;

    task run_phase(uvm_phase phase);
        test_seqh=random_virtual_seq::type_id::create("test_seqh");
        phase.raise_objection(this,"start virtual sequence");
        test_seqh.start(envh.v_seqrh); //driven by virtual sequencer
        phase.drop_objection(this,"Done virtual sequence");
    endtask:run_phase

endclass:test_ram_virt_seq
```

10.UVM Callbacks

10.1 Introduction

Callback mechanism will help us for altering the behaviour of a transactor externally, without modifying its existing implementation.

10.2 Example of a transactor without callbacks

10.2.1 Step 1: Definition of driver

```
class driver extends uvm_component;

   `uvm_component_utils(driver)

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

    virtual task run_phase(uvm_phase phase);
       phase.raise_objection(this);
       repeat(2)
          begin
             $display("driver : started driving the packet .....%d",$time);
             // Logic to drive the packet goes here
             // let's consider that it takes 40 time units to drive the packet.
             #40;
             $display("driver : Finished Driving the packet ...%d",$time);
          end
       phase.drop_objection(this);
    endtask : run_phase

 endclass : driver
```

10.2.2 Step 2: Definition of test to run simulation

```
`include "uvm_macros.svh"
 import uvm_pkg::*;
`include "driver.sv"

 module test;

    initial
       begin
          driver drvr;
          drvr = new("drvr");
          run_test();
       end

endmodule : test
```

10.2.3 Step 3: Expected Output

```
UVM_INFO @0: reporter [RNTST] Running test
Driver: Started Driving the packet ..... 0
Driver: Finished Driving the packet ..... 40
Driver: Started Driving the packet ..... 40
Driver: Finished Driving the packet ..... 80
```

10.3 Example of a transactor with callbacks

10.3.1 Step 1: Definition of driver callback class with empty callback methods

```
class driver_callback extends uvm_callback;

   function new(string name = "driver_callback");
      super.new(name);
   endfunction : new

   static string type_name = "driver_callback";

   virtual function string get_type_name();
      return type_name;
   endfunction : get_type_name

   virtual task pre_send();
   endtask

   virtual task post_send();
   endtask

endclass : driver_callback
```

10.3.2 Step 2: Definition of driver class with empty callback hooks

```
class driver extends uvm_component;
    `uvm_component_utils(driver)
    `uvm_register_cb(driver,driver_callback)

    function new(string name,uvm_component parent);
       super.new(name,parent);
    endfunction : new

    virtual task run_phase(uvm_phase phase);
       phase.raise_objection(this);
       repeat(2)
          begin
             `uvm_do_callbacks(driver,driver_callback,pre_send());

             $display("driver:started driving the packet ....%d"$time);
             //Logic to drive thee packet goes here which takes 40 time units
             #40;
             $display("driver:finished driving the packet ....%d"$time);

             `uvm_do_callbacks(driver,driver_callback,post_send());
          end
       phase.drop_objection(this);

    endtask : run_phase

 endclass : driver
```

10.3.3 Step 3: Definition of extended driver callback class with callback methods

```
class custom_driver_callbacks_1 extends driver_callback;

   function new(string name="driver_callback");
      super.new(name);
   endfunction : new

   virtual task pre_send();
      $display("CB_1:pre_send:Delaying the packet driving by 20 time units. %d ",$time);
      #20;
   endtask : pre_send

   virtual task post_send();
      $display("CB_1:post_send:Just a message from post_send callback method \n");
   endtask : post_send

endclass:custom_driver_callbacks_1
```

10.3.4 Step 4: Linking Driver and Callback classes

```
module test;
   initial
      begin
         driver drvr;
         custom_callbacks_1 cb_1;
         drvr = new("drvr");
         cb_1=new("cb_1");
         uvm_callback #(driver,driver_callback)::add(drvr,cb_1);
         uvm_callback #(driver,driver_callback)::display;
         run_test();
      end
endmodule : test
```

10.3.5 Expected output

```
---------------------------------------------------------
Registered callbacks for all instances of Driver
---------------------------------------------------------
cb_1 on drvr ON
UVM_INFO @0: reporter [RNTST] Running test
CB_1:pre_send: Delaying the packet driving by 20 time units at time t = 0
Driver: Started Driving the packet ..... 20
Driver: Finished Driving the packet ..... 60
CB_1:post_send: Just a message from post send callback method

CB_1:pre_send: Delaying the packet driving by 20 time units at time t = 60
Driver: Started Driving the packet ..... 80
Driver: Finished Driving the packet ..... 120
CB_1:post_send: Just a message from post send callback method
```

11.UVM Events

11.1 Introduction

Events help us to achieve the synchronization between various TB components.

11.2 Example to demonstrate the usage of uvm events

11.2.1 Step 1: Configuration Class with event pool

```
class config_class extends uvm_object;
    `uvm_object_utils(config_class)
    uvm_active_passive_enum is_active = UVM_ACTIVE;
    uvm_event_pool event_pool=uvm_event_pool::get_global_pool();

    function new(string name = "config_class");
        super.new(name);
    endfunction:new

endclass:config_class
```

11.2.2 Step 2: Definition of event callback class

```
class event_cb  extends uvm_event_callback;
    `uvm_object_utils(event_cb)

    function new(string name = "event_cb");
        super.new(name);
    endfunction : new

    function bit pre_trigger(uvm_event e,uvm_object data =null);
        $display("pre trigger function of event callback object :: Time is %0t",$time);
        return(0);
    endfunction : pre_trigger

    function void post_trigger(uvm_event e,uvm_object data =null);
        $display("post trigger function of event callback object :: Time is %0t",$time);
    endfunction : post_trigger

endclass : event_cb
```

11.2.3 Step 3: Definition of Driver with UVM events

```
task driver::dir_task(seq_item req);
    uvm_event e1 = cfg.event_pool.get("e1");
    uvm_event e2 = cfg.event_pool.get("e2");

    e2.add_callback(eve_cb,0);//Regiters the callback object eve_cb with event e2

    for(int i = 0;i<5;i++)
        begin
            vif.adv_pkt_stop<= i;
            #5;
        end
    e1.trigger(); // Triggering thee event e1
    $display("event e1 triggered at time %t in driver ",$time);
    #30;
    e2.trigger();
    $display("event e2 triggered at time %t in driver ",$time);

endtask : dir_task
```

11.2.4 Step 4: Definition of Monitor with UVM events

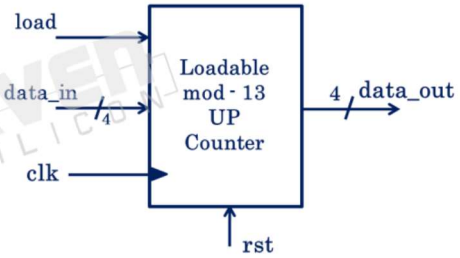
```
task monitor :: run();

   uvm_event e1 = cfg.event_pool.get("e1");
   uvm_event e2 = cfg.event_pool.get("e2");
   fork
      begin
         e1.wait_ptrigger(); //wait for the trigger of event e1 in driver
         $display("Monitor:Trigger of event e1 at time %t",$time);
      end

      begin
         e2.wait_ptrigger(); //wait for the trigger of event e2 in driver
         $display("Monitor:Trigger of event e2 at time %t",$time);
      end

   join
   `uvm_info(get_name(),"PRINTING FROM RUN PHASE OF MONITOR ",UVM_LOW)

endtask:run
```

11.2.5 Expected Output

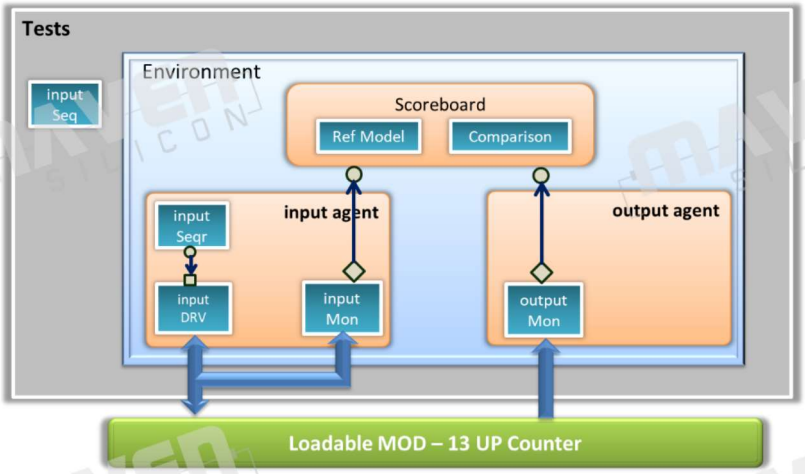
```
event e1 is triggered at time       25 in driver
Monitor:: Trigger of event e1 at time        25
Pre trigger function of event callback object:: Time is 55
Post trigger function of event callback object:: Time is 55
event e2 is triggered at time       55 in driver
Monitor:: Trigger of event e2 at time        55
```

12.Verification of Modulo – 13 Loadable UP counter

12.1 Block Diagram



12.2 TB Architecture



12.3 Interface

```
interface counter_inf(input bit clock);

    logic [3:0] data_in;
    logic [3:0] data_out;
    logic load;
    logic rst;

//Driver Clocking Block
    clocking driver_cb@(posedge clock);
        default input #1 output #1;
        output rst;
        output data_in;
        output load;
    endclocking

//output monitor clocking block
    clocking output_mon_cb@(posedge clock);
        default input #1 output #1;
        input data_out;
    endclocking
```

```
//input monitor clocking block
    clocking input_mon_cb@(posedge clock);
        default input #1 output #1;
        input load;
        input rst;
        input data_in;
    endclocking

//driver modport
    modport DRIVER (clocking driver_cb);

//input Monitor modport
    modport INPUT_MON (clocking input_mon_cb);

//Output monitor modport
    modport OUTPUT_MON (clocking output_mon_cb);

endinterface
```

12.4 Transaction Class (Sequence Item)

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

    rand logic rst;
    rand logic load;
    randc logic [3:0] data_in;
    logic [3:0] data_out;
    static int no_of_xtn;

    constraint VALID_RST {rst dist{1:= 1, 0:= 15};}
    constraint VALID_LOAD {load dist{1:= 1, 0:= 15};}
    constraint VALID_DATA {data_in inside {[0:15]};}

    function new(string name = "counter_trans");
        super.new(name);
    endfunction:new

    function bit do_compare (uvm_object rhs,uvm_comparer comparer);
        counter_trans rhs_;
        if(!$cast(rhs_,rhs))
            begin
                `uvm_fatal("do_compare","cast of the rhs object failed")
                return 0;
            end
        return super.do_compare(rhs,comparer) &&
                        data_out== rhs_.data_out;
    endfunction:do_compare

    function void  do_print (uvm_printer printer);
        super.do_print(printer);
        printer.print_field( "rst", this.rst, '1, UVM_DEC);
        printer.print_field( "load", this.load, '1, UVM_DEC);
        printer.print_field( "data_in", this.data_in, 4, UVM_DEC);
        printer.print_field( "data_out", this.data_out, 4, UVM_DEC);
    endfunction:do_print

    function void post_randomize();
        no_of_xtn++;
        `uvm_info("randomized data", $sformatf("randomized transaction [%0d] is \n%s",
                                                no_of_xtn, this.sprint()), UVM_MEDIUM)
    endfunction:post_randomize
endclass
```

12.5 Configuration Class

```
class counter_env_config extends uvm_object;

    `uvm_object_utils(counter_env_config)

    bit has_input_agent;
    bit has_output_agent;
    bit has_scoreboard;
    uvm_active_passive_enum input_agent_is_active;
    uvm_active_passive_enum output_agent_is_active;
    virtual counter_inf vif;

    function new(string name = "counter_env_config");
        super.new(name);
    endfunction

endclass
```

12.6 Components in the input agent side

12.6.1 Input Driver

```
class counter_input_driver extends uvm_driver #(counter_trans);
    `uvm_component_utils(counter_input_driver)
    virtual counter_inf.DRIVER drv_inf;
    counter_trans data2duv_pkt;
    counter_env_config m_cfg;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(counter_env_config)::get(this,"","counter_env_config",m_cfg))
            `uvm_fatal("CONFIG","cannot get() m_cfg from uvm_config_db. Have you set() it?")
    endfunction

    function void connect_phase(uvm_phase phase);
        drv_inf = m_cfg.vif;
    endfunction

    task run_phase(uvm_phase phase);
        forever
            begin
                seq_item_port.get_next_item(req);
                send_to_dut(req);
                seq_item_port.item_done();
            end
    endtask

    virtual task send_to_dut(counter_trans data2duv_pkt);
        @(drv_inf.driver_cb);
        drv_inf.driver_cb.data_in<=data2duv_pkt.data_in;
        drv_inf.driver_cb.rst<=data2duv_pkt.rst;
        drv_inf.driver_cb.load<=data2duv_pkt.load;
    endtask

endclass
```

12.6.2 Input Sequencer

```
class counter_input_sequencer extends uvm_sequencer #(counter_trans);
    `uvm_component_utils(counter_input_sequencer)

    function new(string name="counter_input_sequencer", uvm_component parent);
        super.new(name,parent);
    endfunction
endclass
```

12.6.3 Input Monitor

```
class counter_input_monitor extends uvm_monitor;
    `uvm_component_utils(counter_input_monitor)

    virtual counter_inf.INPUT_MON mon_if;
    counter_trans drv2mon_pkt;
    counter_env_config m_cfg;

    uvm_analysis_port#(counter_trans) monitor_port;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(counter_env_config)::get(this,
                                        "","counter_env_config",m_cfg))
            `uvm_fatal(get_type_name,"cannot get() m_cfg ")
        monitor_port = new("monitor_port", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        mon_if = m_cfg.vif;
    endfunction

    task run_phase(uvm_phase phase);
        @(mon_if.input_mon_cb);
        forever
            monitor();
    endtask

    task monitor();
        begin
            @(mon_if.input_mon_cb);
            drv2mon_pkt = counter_trans::type_id::create("drv2mon_pkt");
            drv2mon_pkt.load = mon_if.input_mon_cb.load;
            drv2mon_pkt.rst =  mon_if.input_mon_cb.rst;
            drv2mon_pkt.data_in= mon_if.input_mon_cb.data_in;
            `uvm_info(get_type_name, $sformatf("input monitor has captured
                    below transaction \n%s", drv2mon_pkt.sprint()), UVM_MEDIUM)
            monitor_port.write(drv2mon_pkt);

        end
    endtask
endclass
```

12.6.4 Input Agent

```
class counter_input_agent extends uvm_agent;
    `uvm_component_utils(counter_input_agent)

    counter_env_config m_cfg;

    counter_input_monitor monh;
    counter_input_sequencer seqrh;
    counter_input_driver drvh;

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

    function void build_phase(uvm_phase phase);

        super.build_phase(phase);

        if(!uvm_config_db #(counter_env_config)::get(this,"","counter_env_config",m_cfg))
            `uvm_fatal("CONFIG","cannot get() m_cfg from uvm_config_db. Have you set() it?")
        monh=counter_input_monitor::type_id::create("monh",this);
        if(m_cfg.input_agent_is_active==UVM_ACTIVE)
            begin
                drvh=counter_input_driver::type_id::create("drvh",this);
                seqrh=counter_input_sequencer::type_id::create("seqrh",this);
            end

    endfunction

    function void connect_phase(uvm_phase phase);
        if(m_cfg.input_agent_is_active==UVM_ACTIVE)
            begin
                drvh.seq_item_port.connect(seqrh.seq_item_export);
            end
    endfunction

endclass : counter_input_agent
```

12.7 Components in the output agent side

12.7.1 Output Monitor

```
class counter_output_monitor extends uvm_monitor;
    `uvm_component_utils(counter_output_monitor)

    virtual counter_inf.OUTPUT_MON rdmon_inf;
    counter_trans output_mon_pkt;
    counter_env_config m_cfg;
    uvm_analysis_port#(counter_trans) monitor_port;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(counter_env_config)::get(this,
                                                "","counter_env_config",m_cfg))
            `uvm_fatal(get_type_name,"cannot get() m_cfg ")
        monitor_port = new("monitor_port", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        rdmon_inf = m_cfg.vif;
    endfunction

    task run_phase(uvm_phase phase);
        repeat(2)
            @(rdmon_inf.output_mon_cb);
        forever
            monitor();
    endtask

    task monitor();
        output_mon_pkt = counter_trans::type_id::create("output_mon_pkt");
        @(rdmon_inf.output_mon_cb);
        output_mon_pkt.data_out = rdmon_inf.output_mon_cb.data_out;
        `uvm_info(get_type_name, $sformatf("output monitor has captured
                  below transaction \n%s", output_mon_pkt.sprint()), UVM_MEDIUM)
        monitor_port.write(output_mon_pkt);
    endtask

endclass
```

12.7.2 Output Agent

```
class counter_output_agent extends uvm_agent;
    `uvm_component_utils(counter_output_agent)

    counter_env_config m_cfg;

    counter_output_monitor monh;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(counter_env_config)::get(this,"","counter_env_config",m_cfg))
            `uvm_fatal("CONFIG","cannot get() m_cfg from uvm_config_db. Have you set() it?")
        if(m_cfg.output_agent_is_active==UVM_PASSIVE)
            monh=counter_output_monitor::type_id::create("monh",this);
    endfunction
endclass : counter_output_agent
```

12.8 Scoreboard with Reference Model Logic

```
class counter_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(counter_scoreboard)

    uvm_tlm_analysis_fifo #(counter_trans) input_mon_fifo;
    uvm_tlm_analysis_fifo #(counter_trans) output_mon_fifo;

    counter_trans input_mon_pkt;
    counter_trans output_mon_pkt;
    counter_trans expected_output_pkt;
    counter_trans cov_data_pkt;

    int data_verified = 0;
    int input_mon_pkt_count = 0;
    int output_mon_pkt_count = 0;
            covergroup counter_coverage;
        option.per_instance=1;
        LOAD_DATA : coverpoint cov_data_pkt.data_in {
            bins ZERO     = {0};
            bins LOW1     = {[1:2]};
            bins LOW2     = {[3:4]};
            bins MID_LOW  = {[5:6]};
            bins MID      = {[7:8]};
            bins MID_HIGH = {[9:10]};
            bins HIGH1    = {[11:12]};
            bins HIGH2    = {[13:14]};
            bins MAX      = {15};
            }
        RESET_CMD     : coverpoint cov_data_pkt.rst {
            bins cmd_rst = {1};
            }
        LOAD_CMD      : coverpoint cov_data_pkt.load {
            bins load_dut  = {1};
            }
        READxWRITE      : cross LOAD_CMD, LOAD_DATA;
    endgroup

    function new(string name,uvm_component parent);
        super.new(name,parent);
        input_mon_fifo = new("input_mon_fifo", this);
        output_mon_fifo = new("output_mon_fifo", this);
        expected_output_pkt = counter_trans::type_id::create("expected_output_pkt");
        counter_coverage = new;
    endfunction
    task run_phase(uvm_phase phase);
        forever
            begin
                input_mon_fifo.get(input_mon_pkt);
                input_mon_pkt_count++;
                `uvm_info(get_type_name, $sformatf("sb has got below pkt from input monitor \n%s",
                                                    input_mon_pkt.sprint()), UVM_MEDIUM)
                output_mon_fifo.get(output_mon_pkt);
                output_mon_pkt_count++;
                `uvm_info(get_type_name, $sformatf("sb has got below pkt from output monitor \n%s",
                                                    output_mon_pkt.sprint()), UVM_MEDIUM)
                ref_model_logic();
                validate_output();
            end
    endtask
```

Continued in the next page…

```
    task ref_model_logic();
        begin
            if(input_mon_pkt.rst || (input_mon_pkt.load&input_mon_pkt.data_in >= 13))
                expected_output_pkt.data_out = 4'b0;
            else if(input_mon_pkt.load)
                expected_output_pkt.data_out = input_mon_pkt.data_in;
            else if(expected_output_pkt.data_out >= 13)
                expected_output_pkt.data_out = 4'b0;
            else
                expected_output_pkt.data_out = expected_output_pkt.data_out + 1'b1;
        end
    endtask

    virtual task validate_output();
        if(!expected_output_pkt.compare(output_mon_pkt))
            begin:failed_compare
                `uvm_info(get_type_name, $sformatf("expected packet is below\n%s",
                                        expected_output_pkt.sprint()), UVM_MEDIUM)
                `uvm_info(get_type_name, $sformatf("dut's output packet is below\n%s",
                                        output_mon_pkt.sprint()), UVM_MEDIUM)
                //$finish;
            end:failed_compare
        else
            begin
                `uvm_info(get_type_name(), $sformatf("Data Match successful"), UVM_MEDIUM)
                data_verified++ ;
            end
        cov_data_pkt = input_mon_pkt;
        counter_coverage.sample();

    endtask

    function void report_phase(uvm_phase phase);
        $display(" \n----------------------- SCOREBOARD REPORT ---------------------- \n ");
        $display(" input mon pkt count = %0d , output mon pkt count = %0d,
                   no_of_successful comparisions = %0d\n",
                   input_mon_pkt_count,output_mon_pkt_count ,data_verified);
        $display(" ---------------------------------------------------------------- \n ");
    endfunction
endclass
```

12.9 Environment

```
class counter_env extends uvm_env;

    `uvm_component_utils(counter_env)
    counter_input_agent input_agent_h;
    counter_output_agent output_agent_h;
    counter_scoreboard scoreboard_h;

    counter_env_config m_cfg;

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

    function void build_phase(uvm_phase phase);
        if(!uvm_config_db #(counter_env_config)::get(this,"","counter_env_config",m_cfg))
            `uvm_fatal(get_type_name,"cannot get() m_cfg from uvm_config_db. Have you set() it?")
        if(m_cfg.has_input_agent)
            input_agent_h = counter_input_agent::type_id::create("input_agent_h" ,this);
        if(m_cfg.has_output_agent)
            output_agent_h = counter_output_agent::type_id::create("output_agent_h" ,this);
        if(m_cfg.has_scoreboard)
            scoreboard_h = counter_scoreboard::type_id::create("scoreboard_h" ,this);
        super.build_phase(phase);
    endfunction

    function void connect_phase(uvm_phase phase);
        uvm_top.print_topology();
        if(m_cfg.has_input_agent && m_cfg.has_scoreboard)
            input_agent_h.monh.monitor_port.connect(scoreboard_h.input_mon_fifo.analysis_export);
        if(m_cfg.has_output_agent && m_cfg.has_scoreboard)
            output_agent_h.monh.monitor_port.connect(scoreboard_h.output_mon_fifo.analysis_export);
    endfunction

endclass
```

12.10 Sequence

```
class counter_seq extends uvm_sequence #(counter_trans);

    `uvm_object_utils(counter_seq)
    int xtn_num = 1;
    function new(string name = "counter_seq");
        super.new(name);
    endfunction

    task body();
        repeat(number_of_transactions)
            begin
                req=counter_trans::type_id::create("req");
                start_item(req);
                if(xtn_num == 1)
                    begin
                        assert(req.randomize() with {rst == 1;});
                        xtn_num++;
                    end
                else
                    assert(req.randomize());
                finish_item(req);
            end
    endtask

endclass
```

12.11 Base testcase

```
import counter_pkg::*;

class counter_base_test extends uvm_test;
    `uvm_component_utils(counter_base_test)

    counter_env_config m_cfg;

    counter_env counter_env_h;

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


    function void build_phase(uvm_phase phase);
        m_cfg = counter_env_config::type_id::create("m_cfg");
        if(!uvm_config_db #(virtual counter_inf)::get(this,"", "vif",m_cfg.vif))
            `uvm_fatal(get_type_name,"cannot get()interface vif from uvm_config_db.");
        m_cfg.has_input_agent = 1;
        m_cfg.has_output_agent = 1;
        m_cfg.has_scoreboard = 1;
        m_cfg.output_agent_is_active = UVM_PASSIVE;
        m_cfg.input_agent_is_active = UVM_ACTIVE;

        uvm_config_db #(counter_env_config)::set(this,"*","counter_env_config",m_cfg);
        counter_env_h = counter_env::type_id::create("counter_env_h", this);
    endfunction
endclass
```

12.12 Derived testcase

```
class counter_test_1 extends counter_base_test;
    `uvm_component_utils(counter_test_1)

    counter_seq counter_seq_h;

    function new(string name = "counter_test_1",
                uvm_component parent = null);
        super.new(name, parent);
    endfunction


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

    task run_phase(uvm_phase phase);
        phase.raise_objection(this);

        counter_seq_h=counter_seq::type_id::create("counter_seq_h");

        counter_seq_h.start(counter_env_h.input_agent_h.seqrh);
        #30;
        phase.drop_objection(this);
endtask
endclass
```

12.13 Package

```
package counter_pkg;

    int number_of_transactions=100;
    import uvm_pkg::*;

    `include "uvm_macros.svh"

    `include "counter_trans.sv"
    `include "counter_env_config.sv"
    `include "counter_input_driver.sv"
    `include "counter_input_monitor.sv"
    `include "counter_input_sequencer.sv"
    `include "counter_input_agent.sv"
    `include "counter_output_monitor.sv"
    `include "counter_output_agent.sv"
    `include "counter_scoreboard.sv"
    `include "counter_env.sv"
    `include "counter_sequence.sv"
    `include "counter_test.sv"

endpackage
```

12.14 Top Module

```
module counter_top();

    import counter_pkg::*;
    import uvm_pkg::*;

    parameter cycle = 10;

    reg clk;
    counter_inf DUT_INF(clk);

    counter_load DUV(.clk(clk),
                     .rst(DUT_INF.rst),
                     .load(DUT_INF.load),
                     .data_out(DUT_INF.data_out),
                     .data_in(DUT_INF.data_in));

initial
    begin
        uvm_config_db #(virtual counter_inf)::set(null,"*","vif",DUT_INF);
        run_test();
    end

//Generate the clock
    initial
        begin
            clk=1'b0;
            forever
                #(cycle/2) clk=~clk;
        end

endmodule
```