SYSTEMVERILOG-BASED COMPREHENSIVE GUIDE TO VERIFYING THE SWITCH RTL CORE

STEP-BY-STEP TUTORIAL

JAIRAJ MIRASHI
DESIGN VERIFICATION ENGINEER

INTRODUCTION

In this tutorial, we will verify the Switch RTL core. Following are the steps we follow to verify the Switch RTL core.

1) Understand the specification

2) Developing Verification Plan

3) Building the Verification Environment. We will build the Environment in Multiple phases, so it will be easy for you to lean step by step.

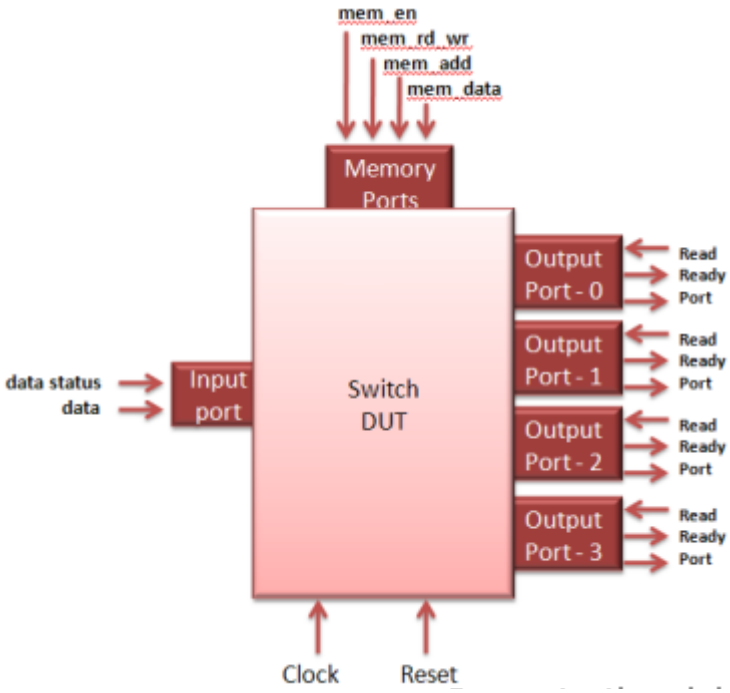
- Phase 1) We will develop the testcase and interfaces, and integrate them in these with the DUT in top module.
- Phase 2) We will Develop the Environment class.
- Phase 3) We will develop reset and configuration methods in Environment class. Then using these methods, we will reset the DUT and configure the port address.
- Phase 4) We will develop a packet class based on the stimulus plan. We will also write a small code to test the packet class implementation.
- Phase 5) We will develop a driver class. Packets are generated and sent to dut using driver.
- Phase 6) We will develop receiver class. Receiver collects the packets coming from the output port of the DUT.
- Phase 7) We will develop scoreboard class which does the comparison of the expected packet with the actual packet received from the DUT.
- Phase 8) We will develop coverage class based on the coverage plan.
- Phase 9) In this phase , we will write testcases and analyze the coverage report.

SPECIFICATION

Switch Specification:

This is a simple switch. Switch is a packet based protocol. Switch drives the incoming packet which comes from the input port to output ports based on the address contained in the packet.

The switch has a one input port from which the packet enters. It has four output ports where the packet is driven out.



Packet Format:

Packet contains 3 parts. They are Header, data and frame check sequence.
Packet width is 8 bits and the length of the packet can be between 4 bytes to 259 bytes.

Packet Header:

Packet header contains three fields DA, SA and length.

- DA: Destination address of the packet is of 8 bits. The switch drives the packet to respective ports based on this destination address of the packets. Each output port has 8-bit unique port address. If the destination address of the packet matches the port address, then switch drives the packet to the output port.

- SA: Source address of the packet from where it originate. It is 8 bits.

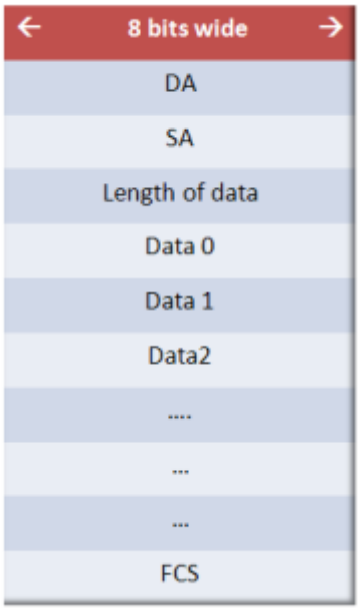
- Length: Length of the data is of 8 bits and from 0 to 255. Length is measured in terms of bytes.

If Length = 0, it means data length is 0 bytes
If Length = 1, it means data length is 1 bytes
If Length = 2, it means data length is 2 bytes
If Length = 255, it means data length is 255 bytes

- Data: Data should be in terms of bytes and can take anything.

- FCS: Frame check sequence

This field contains the security check of the packet. It is calculated over the header and data.



Configuration:

Switch has four output ports. These output ports address have to be configured to a unique address. Switch matches the DA field of the packet with this configured port address and sends the packet on to that port. Switch contains a memory. This memory has 4 locations, each can store 8 bits. To configure the switch port address, memory write operation has to be done using memory interface. Memory address (0,1,2,3) contains the address of port(0,1,2,3) respectively.

Interface Specification:

The Switch has one input Interface, from where the packet enters and 4 output interfaces from where the packet comes out and one memory interface, through the port address can be configured. Switch also has a clock and asynchronous reset signal.

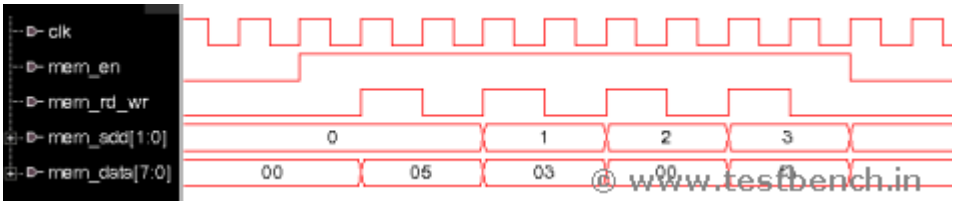
Memory Interface:

Through memory interfaced output port address are configured. It accepts 8 bit data to be written to memory. It has 8 bit address inputs. Address 0,1,2,3 contains the address of the port 0,1,2,3 respectively.

There are 4 input signals to memory interface. They are

input mem_en;
input mem_rd_wr;
input[1:0] mem_add;
input[7:0] mem_data;

All the signals are active high and are synchronous to the positive edge of clock signal.
To configure a port address,
1. Assert the mem_en signal.
2. Asser the mem_rd_wr signal.
3. Drive the port number (0 or 1 or 2 or 3) on the mem_add signal
4. Drive the 8 bit port address on to mem_data signal.



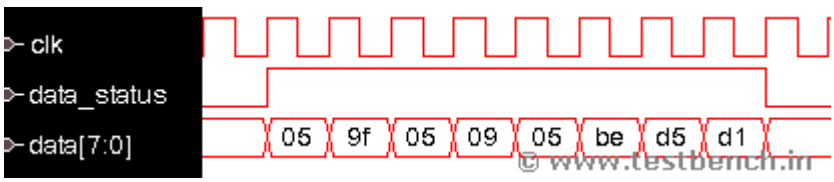
Input Port

Packets are sent into the switch using input port.
All the signals are active high and are synchronous to the positive edge of clock signal.

input port has 2**input** signals. They are
input[7:0]data;
input data_status;

To send the packet in to switch,

1. Assert the data_status signal.
2. Send the packet on the data signal byte by byte.
3. After sending all the data bytes, deassert the data_status signal.
4. There should be at least 3 clock cycles difference between packets.



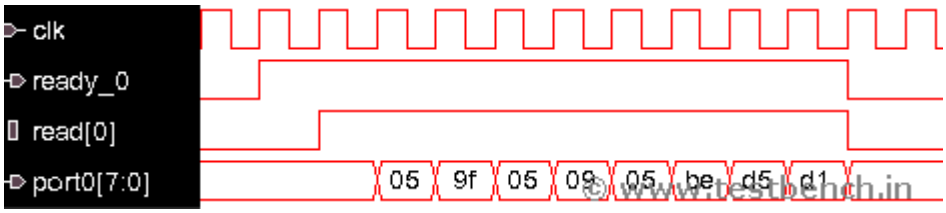
Output Port

Switch sends the packets out using the output ports. There are 4 ports, each having data, ready and read signals. All the signals are active high and are synchronous to the positive edge of clock signal.

Signal list is

output[7:0] port0;
output[7:0] port1;
output[7:0] port2;
output[7:0] port3;
output ready_0;
output ready_1;
output ready_2;
output ready_3;
input read_0;
input read_1;
input read_2;
input read_3;

When the data is ready to be sent out from the port, switch asserts ready_* signal high indicating that data is ready to be sent.
If the read_* signal is asserted, when ready_* is high, then the data comes out of the port_* signal after one clock cycle.



(S) RTL code:

RTL code is attached with the tar files. From the Phase 1, you can download the tar files.

VERIFICATION PLAN

Overview

This Document describes the Verification Plan for Switch. The Verification Plan is based on System Verilog Hardware Verification Language. The methodology used for Verification is Constraint random coverage driven verification.

Feature Extraction

This section contains list of all the features to be verified.
1)
ID: Configuration
Description: Configure all the 4 port address with unique values.

2)
ID: Packet DA
Description: DA field of packet should be any of the port address. All the 4 port address should be used.

3)
ID : Packet payload
Description: Length can be from 0 to 255. Send packets with all the lengths.

4)
ID: Length
Description:
Length field contains length of the payload.
Send Packet with correct length field and incorrect length fields.

5)
ID: FCS
Description:
Good FCS: Send packet with good FCS.
Bad FCS: Send packet with corrupted FCS.

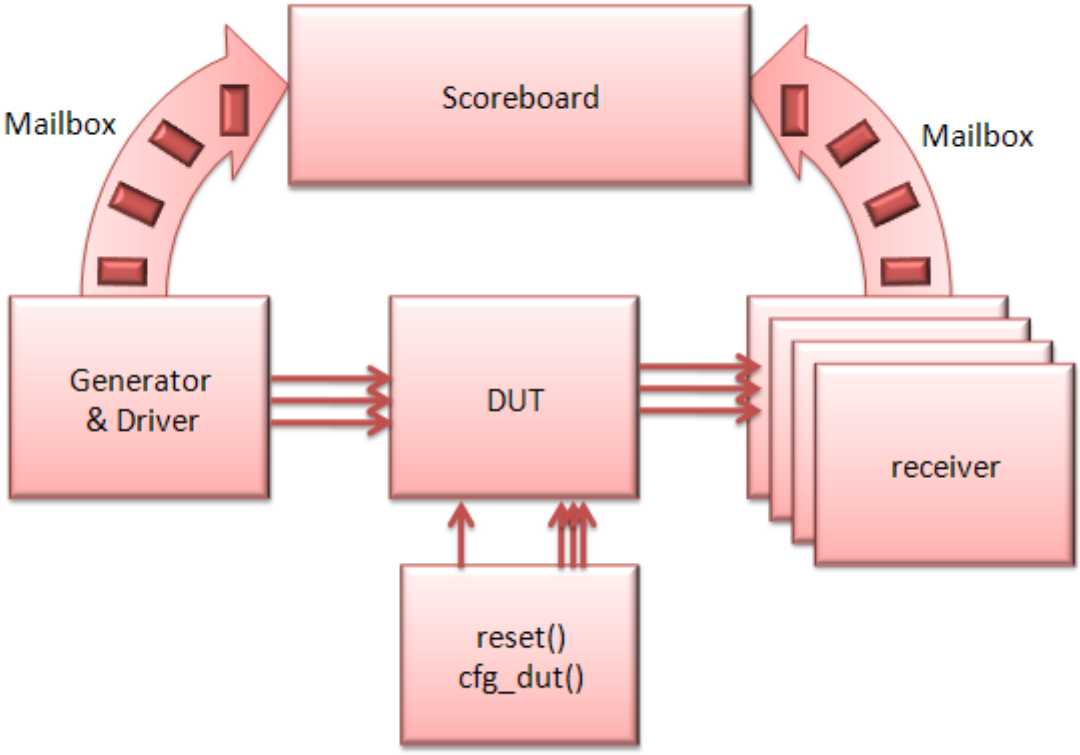
Stimulus Generation Plan

1) Packet DA: Generate packet DA with the configured address.
2) Payload length: generate payload length ranging from 2 to 255.
3) Correct or Incorrect Length field.
4) Generate good and bad FCS.

Coverage Plan

1) Cover all the port address configurations.
2) Cover all the packet lengths.
3) Cover all correct and incorrect length fields.
4) Cover good and bad FCS.
5) Cover all the above combinations.

Verification Environment



Report a Bug or Comment on This section - Your input is what keeps Testbench.in improving with time!

PHASE 1 TOP

In phase 1,

1) We will write SystemVerilog Interfaces for input port, output port and memory port.
2) We will write Top module where testcase and DUT instances are done.
3) DUT and TestBench interfaces are connected in top module.
4) Clock is generator in top module.

NOTE: In every file you will see the syntax
`ifndefGUARD_*
`endifGUARD_*

Interfaces

In the interface.sv file, declare the 3 interfaces in the following way.

- All the interfaces has clock as input.
- All the signals in interface are logic type.
- All the signals are synchronized to clock except reset in clocking block.
- Signal directional w.r.t TestBench is specified with modport.

```
`ifndefGUARD_INTERFACE
`defineGUARD_INTERFACE

////////////////////////////////////////
// Interface declaration for the memory///
////////////////////////////////////////

interface mem_interface(inputbit clock);
logic[7:0] mem_data;
logic[1:0] mem_add;
logic mem_en;
logic mem_rd_wr;

clocking cb@(posedge clock);
defaultinput#1output#1;
output mem_data;
output mem_add;
output mem_en;
output mem_rd_wr;
endclocking

modportMEM(clocking cb,input clock);
```

```
endinterface

//////////////////////////////////////////
// Interface for the input side of switch.//
// Reset signal is also passed hear. //
//////////////////////////////////////////
interface input_interface(inputbit clock);
logic data_status;
logic[7:0] data_in;
logic reset;

clocking cb@(posedge clock);
defaultinput#1output#1;
output data_status;
output data_in;
endclocking

modportIP(clocking cb,output reset,input clock);

endinterface

////////////////////////////////////////////////
// Interface for the output side of the switch.//
// output_interface is for only one output port//
////////////////////////////////////////////////

interface output_interface(inputbit clock);
logic[7:0] data_out;
logic ready;
logic read;

clocking cb@(posedge clock);
defaultinput#1output#1;
input data_out;
input ready;
output read;
endclocking

modportOP(clocking cb,input clock);

endinterface

////////////////////////////////////////////////

`endif
```

Testcase

Testcase is a program block which provides an entry point for the test and creates a scope that encapsulates program-wide data. Currently this is an empty testcase which just ends the simulation after 100 time units. Program block contains all the above declared interfaces as arguments. This testcase has initial and final blocks.

```
`ifndefGUARD_TESTCASE
`defineGUARD_TESTCASE

program testcase(mem_interface.MEM mem_intf,input_interface.IP input_intf,output_interface.OP output_intf[4]);

initial
begin
$display(" ******************* Start of testcase *****************");

#1000;
end

final
$display(" ******************* End of testcase ******************");

endprogram
`endif
```

Top Module

The modules that are included in the source text but are not instantiated are called top modules. This module is the highest scope of modules. Generally this module is named as "top" and referenced as "top module". Module name can be anything.

Do the following in the top module:

1)Generate the clock signal.

```
bit Clock;

initial
forever#10 Clock =~Clock;
```

2)Do the instances of memory interface.

```
mem_interface mem_intf(Clock);
```

3)Do the instances of input interface.

```
input_interface input_intf(Clock);
```

4)There are 4 output ports. So do 4 instances of output_interface.

```
output_interface output_intf[4](Clock);
```

5)Do the instance of testcase and pass all the above declared interfaces.

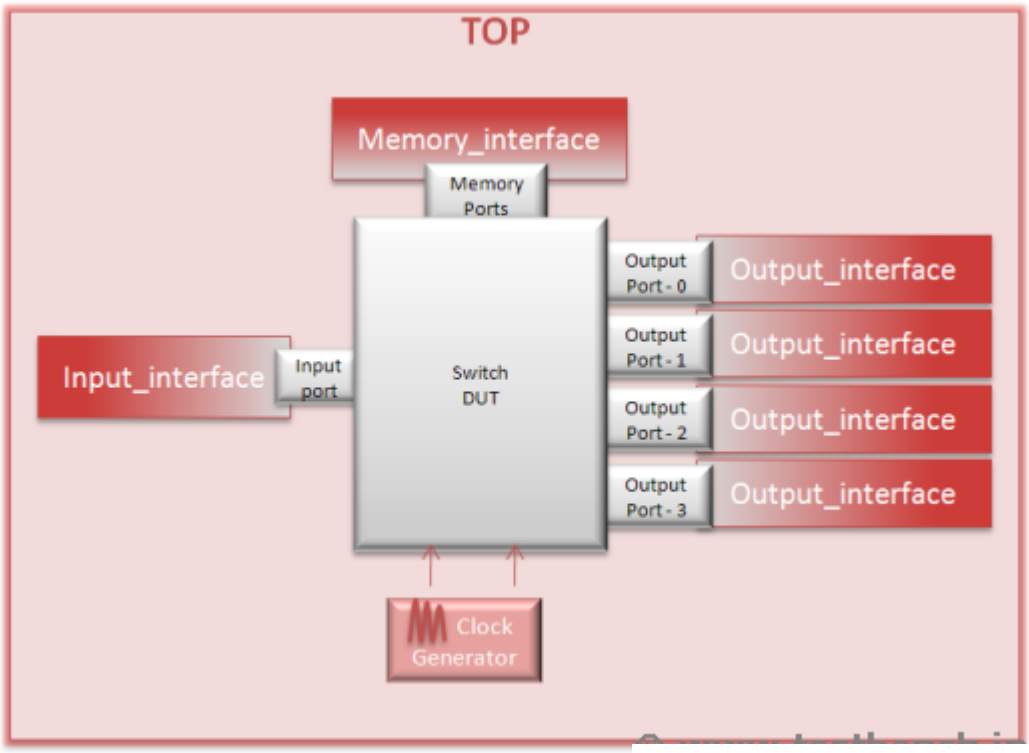
testcase TC(mem_intf,input_intf,output_intf);

6)Do the instance of DUT.

switch DUT(.

7)Connect all the interfaces and DUT. The design which we have taken is in verilog. So Verilog DUT instance is connected signal by signal.

```
switch DUT(.clk(Clock),
.reset(input_intf.reset),
.data_status(input_intf.data_status),
.data(input_intf.data_in),
.port0(output_intf[0].data_out),
.port1(output_intf[1].data_out),
.port2(output_intf[2].data_out),
.port3(output_intf[3].data_out),
.ready_0(output_intf[0].ready),
.ready_1(output_intf[1].ready),
.ready_2(output_intf[2].ready),
.ready_3(output_intf[3].ready),
.read_0(output_intf[0].read),
.read_1(output_intf[1].read),
.read_2(output_intf[2].read),
.read_3(output_intf[3].read),
.mem_en(mem_intf.mem_en),
.mem_rd_wr(mem_intf.mem_rd_wr),
.mem_add(mem_intf.mem_add),
.mem_data(mem_intf.mem_data));
```



Top Module Source Code:

```
`ifndefGUARD_TOP
`defineGUARD_TOP

module top();

////////////////////////////////////////////////////
// Clock Declaration and Generation //
////////////////////////////////////////////////////
bit Clock;

initial
forever#10 Clock =~Clock;

////////////////////////////////////////////////////
// Memory interface instance //
////////////////////////////////////////////////////

mem_interface mem_intf(Clock);

////////////////////////////////////////////////////
// Input interface instance //
////////////////////////////////////////////////////

input_interface input_intf(Clock);
```

```
//////////////////////////////////////////////////
// output interface instance //
//////////////////////////////////////////////////

output_interface output_intf[4](Clock);

//////////////////////////////////////////////////
// Program block Testcase instance //
//////////////////////////////////////////////////

testcase TC(mem_intf,input_intf,output_intf);

//////////////////////////////////////////////////
// DUT instance and signal connection //
//////////////////////////////////////////////////

switch DUT(.clk(Clock),
.reset(input_intf.reset),
.data_status(input_intf.data_status),
.data(input_intf.data_in),
.port0(output_intf[0].data_out),
.port1(output_intf[1].data_out),
.port2(output_intf[2].data_out),
.port3(output_intf[3].data_out),
.ready_0(output_intf[0].ready),
.ready_1(output_intf[1].ready),
.ready_2(output_intf[2].ready),
.ready_3(output_intf[3].ready),
.read_0(output_intf[0].read),
.read_1(output_intf[1].read),
.read_2(output_intf[2].read),
.read_3(output_intf[3].read),
.mem_en(mem_intf.mem_en),
.mem_rd_wr(mem_intf.mem_rd_wr),
.mem_add(mem_intf.mem_add),
.mem_data(mem_intf.mem_data));

endmodule

`endif
```

(S)Download the phase 1 files:

switch_1.tar

(S)Run the simulation:

```
vcs -sverilog -f filelist -R -ntb_opts dtm
```

(S)Log file after simulation:

********** Start of testcase **********
********** End of testcase **********

PHASE 2 ENVIRONMENT

In this phase, we will write

- Environment class.
- Virtual interface declaration.
- Defining Environment class constructor.
- Defining required methods for execution. Currently these methods will not be implemented in this phase.

All the above are done in Environment.sv file.

We will write a testcase using the above define environment class in testcase.sv file.

Environment Class:

The class is a base class used to implement verification environments. Testcase contains the instance of the environment class and has access to all the public declaration of environment class.

All methods are declared as virtual methods. In environment class, we will formalize the simulation steps using virtual methods. The methods are used to control the execution of the simulation.

Following are the methods which are going to be defined in environment class.

1) new() : In constructor method, we will connect the virtual interfaces which are passed as argument to the virtual interfaces to those which are declared in environment class.

2) build(): In this method , all the objects like driver, monitor etc are constructed. Currently this method is empty as we did not develop any other component.

3) reset(): in this method we will reset the DUT.

4) cfg_dut(): In this method, we will configure the DUT output port address.

5) start(): in this method, we will call the methods which are declared in the other components like driver and monitor.

6) wait_for_end(): this method is used to wait for the end of the simulation. Waits until all the required operations in other components are done.

7) report(): This method is used to print the TestPass and TestFail status of the simulation, based on the error count..

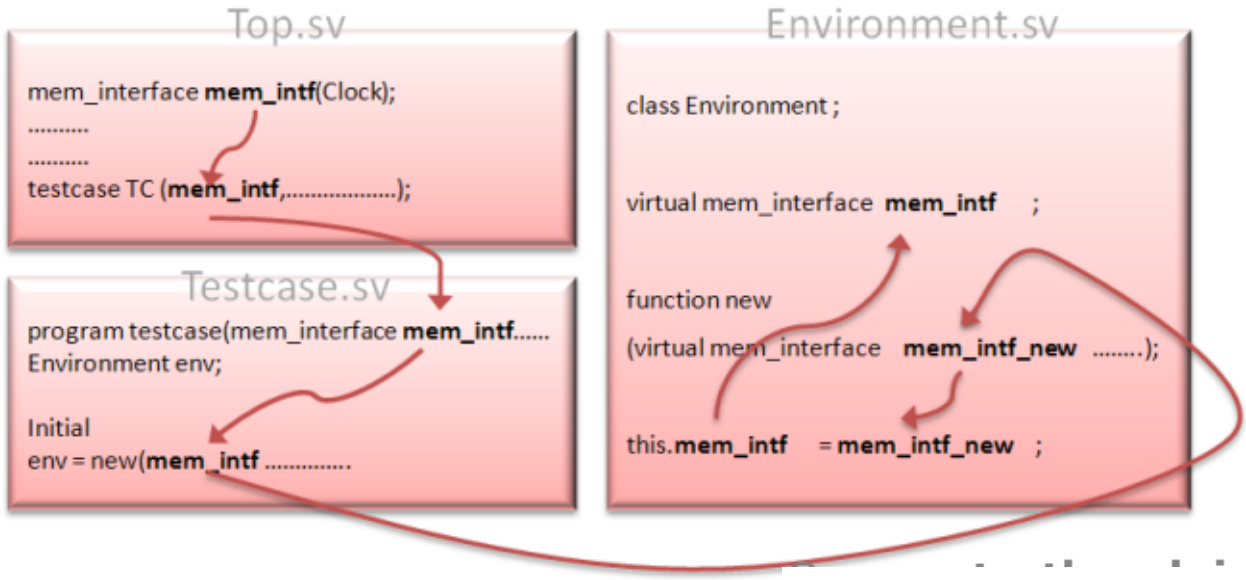
8) run(): This method calls all the above declared methods in a sequence order. The testcase calls this method, to start the simulation.

We are not implementing build(), reset(), cfg_dut() , strat() and report() methods in this phase.

Connecting the virtual interfaces of Environment class to the physical interfaces of top module.

Verification environment contains the declarations of the virtual interfaces. Virtual interfaces are just a handles(like pointers). When a virtual interface is declared, it only creats a handle. It doesnot creat a real interface.

Constructor method should be declared with virtual interface as arguments, so that when the object is created in testcase, new() method can pass the interfaces in to environment class where they are assigned to the local virtual interface handle. With this, the Environment class virtual interfaces are pointed to the physical interfaces which are declared in the top module.



Declare virtual interfaces in Environment class.

virtual mem_interface.MEM mem_intf ;
virtual input_interface.IP input_intf ;
virtual output_interface.OP output_intf[4];

The construction of Environment class is declared with virtual interface as arguments.

functionnew(**virtual** mem_interface.MEM mem_intf_new ,
virtual input_interface.IP input_intf_new ,
virtual output_interface.OP output_intf_new[4]);

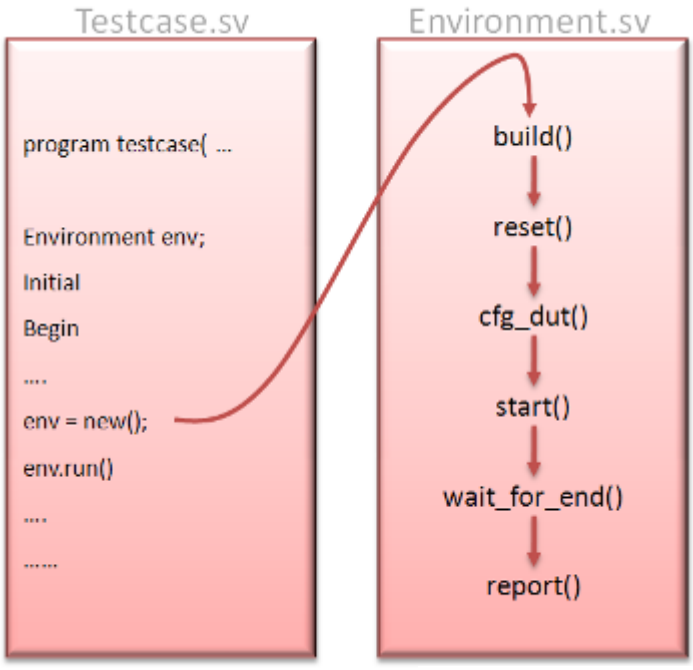
In constructor methods, the interfaces which are arguments are connected to the virtual interfaces of environment class.

this.mem_intf = mem_intf_new ;
this.input_intf = input_intf_new ;
this.output_intf = output_intf_new ;

Run :

The run() method is called from the testcase to start the simulation. run() method calls all the methods which are defined in the Environment class.

task run();
$display(" %0d : Environment : start of run() method",$time);
build();
reset();
cfg_dut();
start();
wait_for_end();
report();
$display(" %0d : Environment : end of run() method",$time);
endtask: run



Environment Class Source Code:

`ifndefGUARD_ENV
`defineGUARD_ENV

class Environment ;

```
virtual mem_interface.MEM mem_intf ;
virtual input_interface.IP input_intf ;
virtual output_interface.OP output_intf[4];

functionnew(virtual mem_interface.MEM mem_intf_new ,
virtual input_interface.IP input_intf_new ,
virtual output_interface.OP output_intf_new[4]);

this.mem_intf = mem_intf_new ;
this.input_intf = input_intf_new ;
this.output_intf = output_intf_new ;

$display(" %0d : Environment : created env object",$time);
endfunction:new

functionvoid build();
$display(" %0d : Environment : start of build() method",$time);
$display(" %0d : Environment : end of build() method",$time);
endfunction:build

task reset();
$display(" %0d : Environment : start of reset() method",$time);
$display(" %0d : Environment : end of reset() method",$time);
endtask: reset

task cfg_dut();
$display(" %0d : Environment : start of cfg_dut() method",$time);
$display(" %0d : Environment : end of cfg_dut() method",$time);
endtask: cfg_dut

taskstart();
$display(" %0d : Environment : start of start() method",$time);
$display(" %0d : Environment : end of start() method",$time);
endtask:start

task wait_for_end();
$display(" %0d : Environment : start of wait_for_end() method",$time);
$display(" %0d : Environment : end of wait_for_end() method",$time);
endtask: wait_for_end

task run();
$display(" %0d : Environment : start of run() method",$time);
build();
reset();
cfg_dut();
start();
wait_for_end();
```

```
report();
$display(" %0d : Environment : end of run() method",$time);
endtask: run

task report();
endtask: report

endclass

`endif
```

We will create a file Global.sv for global requirement. In this file, define all the port address as macros in this file. Define a variable error as integer to keep track the number of errors occurred during the simulation.

```
`ifndefGUARD_GLOBALS
`defineGUARD_GLOBALS

`defineP08'h00
`defineP18'h11
`defineP28'h22
`defineP38'h33

int error =0;
int num_of_pkts =10;

`endif
```

Now we will update the testcase. Take an instance of the Environment class and call the run method of the Environment class.

```
`ifndef GUARD_TESTCASE
`define GUARD_TESTCASE

program testcase(mem_interface.MEM mem_intf,input_interface.IP input_intf,output_interface.OP output_intf[4]);

Environment env;

initial
begin
$display(" ****************** Start of testcase ****************");

env =new(mem_intf,input_intf,output_intf);
env.run();

#1000;
end
```

final
$display(" **************** End of testcase ****************");**

endprogram
`endif

(S)Download the phase 2 source code:

switch_2.tar

(S)Run the simulation:
vcs -sverilog -f filelist -R -ntb_opts dtm

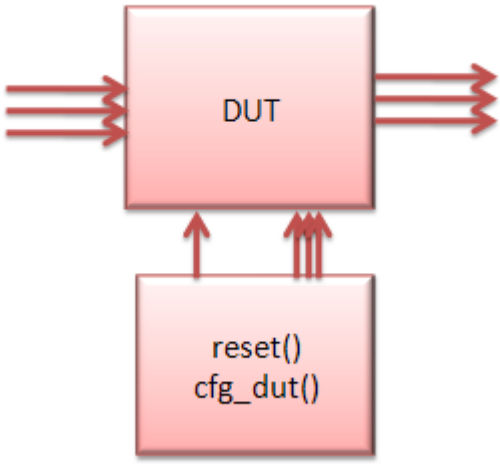
(S)Log report after the simulation:

****************** Start of testcase ****************
0 : Environemnt : created env object
0 : Environemnt : start of run() method
0 : Environemnt : start of build() method
0 : Environemnt : end of build() method
0 : Environemnt : start of reset() method
0 : Environemnt : end of reset() method
0 : Environemnt : start of cfg_dut() method
0 : Environemnt : end of cfg_dut() method
0 : Environemnt : start of start() method
0 : Environemnt : end of start() method
0 : Environemnt : start of wait_for_end() method
0 : Environemnt : end of wait_for_end() method
0 : Environemnt : end of run() method
****************** End of testcase *****************

PHASE 3 RESET

In this phase we will reset and configure the DUT.

The Environment class has reset() method which contains the logic to reset the DUT and cfg_dut() method which contains the logic to configure the DUT port address.



NOTE: Clocking block signals can be driven only using a non-blocking assignment.

Define the reset() method.
1) Set all the DUT input signals to a known state.

```
mem_intf.cb.mem_data <=0;
mem_intf.cb.mem_add <=0;
mem_intf.cb.mem_en <=0;
mem_intf.cb.mem_rd_wr <=0;
input_intf.cb.data_in <=0;
input_intf.cb.data_status <=0;
output_intf[0].cb.read <=0;
output_intf[1].cb.read <=0;
output_intf[2].cb.read <=0;
output_intf[3].cb.read <=0;
```

2) Reset the DUT.

```
// Reset the DUT
input_intf.reset <=1;
repeat(4)@ input_intf.clock;
input_intf.reset <=0;
```

3) Updated the cfg_dut method.

```
task cfg_dut();
$display(" %0d : Environment : start of cfg_dut() method",$time);

mem_intf.cb.mem_en <=1;
@(posedge mem_intf.clock);
mem_intf.cb.mem_rd_wr <=1;

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <=8'h0;
mem_intf.cb.mem_data <=`P0;
$display(" %0d : Environment : Port 0 Address %h ",$time,`P0);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <=8'h1;
mem_intf.cb.mem_data <=`P1;
$display(" %0d : Environment : Port 1 Address %h ",$time,`P1);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <=8'h2;
mem_intf.cb.mem_data <=`P2;
$display(" %0d : Environment : Port 2 Address %h ",$time,`P2);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <=8'h3;
mem_intf.cb.mem_data <=`P3;
$display(" %0d : Environment : Port 3 Address %h ",$time,`P3);

@(posedge mem_intf.clock);
mem_intf.cb.mem_en <=0;
mem_intf.cb.mem_rd_wr <=0;
mem_intf.cb.mem_add <=0;
mem_intf.cb.mem_data <=0;

$display(" %0d : Environment : end of cfg_dut() method",$time);
endtask: cfg_dut
```

(4) In wait_for_end method, wait for some clock cycles.

```
task wait_for_end();
$display(" %0d : Environment : start of wait_for_end() method",$time);
repeat(10000)@(input_intf.clock);
$display(" %0d : Environment : end of wait_for_end() method",$time);
endtask: wait_for_end
```

(S)Download the Phase 3 source code:

switch_3.tar

(S)Run the simulation:

vcs -sverilog -f filelist -R -ntb_opts dtm

(S)Log File report

******************** Start of testcase ******************
0 : Environment : created env object
0 : Environment : start of run() method
0 : Environment : start of build() method
0 : Environment : end of build() method
0 : Environment : start of reset() method
40 : Environment : end of reset() method
40 : Environment : start of cfg_dut() method
70 : Environment : Port 0 Address 00
90 : Environment : Port 1 Address 11
110 : Environment : Port 2 Address 22
130 : Environment : Port 3 Address 33
150 : Environment : end of cfg_dut() method
150 : Environment : start of start() method
150 : Environment : end of start() method
150 : Environment : start of wait_for_end() method
100150 : Environment : end of wait_for_end() method
100150 : Environment : end of run() method
******************** End of testcase *******************

PHASE 4 PACKET

In this Phase, We will define a packet and then test it whether it is generating as expected.

Packet is modeled using class. Packet class should be able to generate all possible packet types randomly. Packet class should also implement required methods like packing(), unpacking(), compare() and display() methods.

We will write the packet class in packet.sv file. Packet class variables and constraints have been derived from stimulus generation plan.

Revisit Stimulus Generation Plan
1) Packet DA: Generate packet DA with the configured address.
2) Payload length: generate payload length ranging from 2 to 255.
3) Correct or Incorrect Length field.
4) Generate good and bad FCS.

1) Declare FCS types as enumerated data types. Name members as GOOD_FCS and BAD_FCS.

typedefenum{GOOD_FCS,BAD_FCS} fcs_kind_t;

2) Declare the length type as enumerated data type. Name members as GOOD_LENGTH and BAD_LENGTH.

typedefenum{GOOD_LENGTH,BAD_LENGTH} length_kind_t;

3) Declare the length type and fcs type variables as rand.

rand fcs_kind_t fcs_kind;
rand length_kind_t length_kind;

4) Declare the packet field as rand. All fields are bit data types. All fields are 8 bit packet array. Declare the payload as dynamic array.

randbit[7:0] length;
randbit[7:0] da;
randbit[7:0] sa;
randbytedata[];//Payload using Dynamic array,size is generated on the fly
randbyte fcs;

5) Constraint the DA field to be any one of the configured address.

constraint address_c { da **inside**{`P0,`P1,`P2,`P3};}

6) Constrain the payload dynamic array size to between 1 to 255.

constraint payload_size_c {data.size**inside**{[1:255]};}

7) Constrain the payload length to the length field based on the length type.

constraint length_kind_c {
(length_kind ==GOOD_LENGTH)-> length ==data.size;
(length_kind ==BAD_LENGTH)-> length ==data.size+2;}

Use solve before to direct the randomization to generate first the payload dynamic array size and then randomize length field.

constraint solve_size_length {**solve**data.size**before** length;}

8) Constrain the FCS field initial value based on the fcs kind field.

constraint fcs_kind_c {
(fcs_kind ==GOOD_FCS)-> fcs ==8'b0;
(fcs_kind ==BAD_FCS)-> fcs ==8'b1;}

9) Define the FCS method.

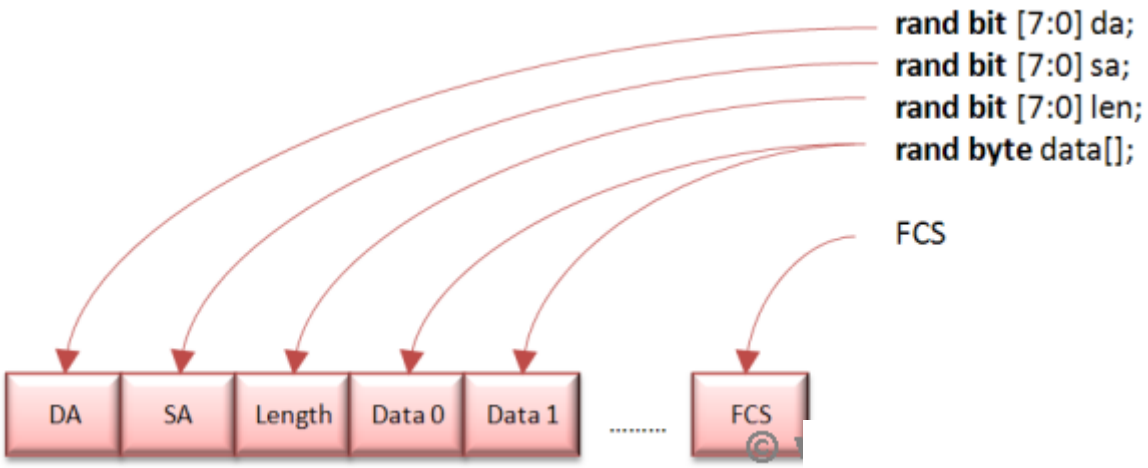
functionbyte cal_fcs;
integer i;
byte result ;
result =0;
result = result ^ da;
result = result ^ sa;
result = result ^ length;
for(i =0;i<data.size;i++)
result = result ^data[i];
result = fcs ^ result;
return result;
endfunction: cal_fcs

10) Define display methods:
Display method displays the current value of the packet fields to standard output.

virtualfunctionvoid display();
$display("\n---------- PACKET KIND ---------- ");
$display(" fcs_kind : %s ",fcs_kind.name());
$display(" length_kind : %s ",length_kind.name());
$display("-------- PACKET ---------- ");
$display(" 0 : %h ",da);
$display(" 1 : %h ",sa);

```
$display(" 2 : %h ",length);
foreach(data[i])
$write("%3d : %0h ",i +3,data[i]);
$display("\n %2d : %h ",data.size()+3, cal_fcs);
$display("------------------------------------------------------------ \n");
endfunction: display
```

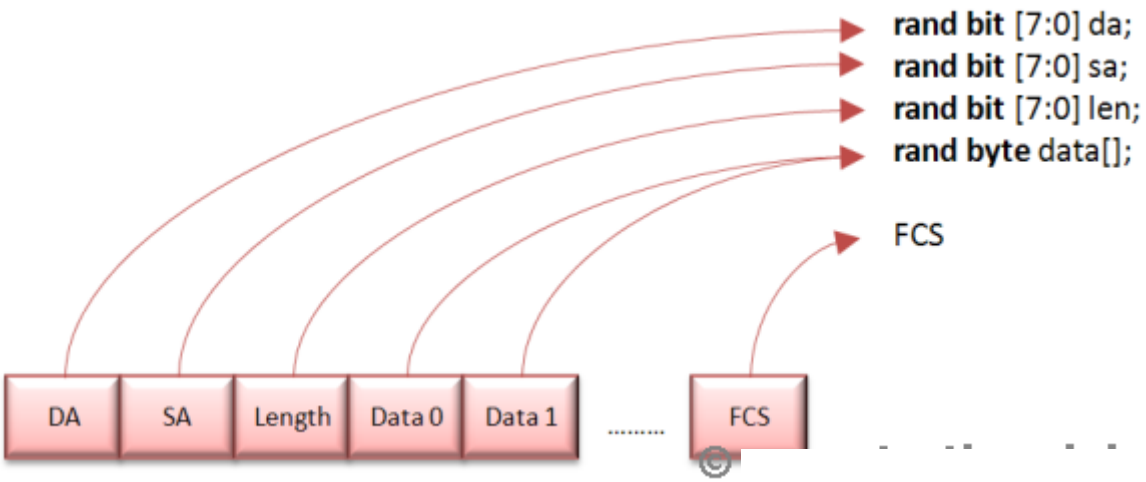
11) Define pack method:



Packing is commonly used to convert the high level data to low level data that can be applied to DUT. In packet class various fields are generated. Required fields are concatenated to form a stream of bytes which can be driven conveniently to DUT interface by the driver.

```
virtualfunctionintunsigned byte_pack(reflogic[7:0] bytes[]);
bytes =new[data.size+4];
bytes[0]= da;
bytes[1]= sa;
bytes[2]= length;
foreach(data[i])
bytes[3+ i]=data[i];
bytes[data.size()+3]= cal_fcs;
byte_pack = bytes.size;
endfunction: byte_pack
```

12) Define unpack method:



The unpack() method does exactly the opposite of pack method. Unpacking is commonly used to convert a data stream coming from DUT to high level data packet object.

```
virtualfunctionvoid byte_unpack(constreflogic[7:0] bytes[]);
this.da = bytes[0];
this.sa = bytes[1];
this.length = bytes[2];
this.fcs = bytes[bytes.size-1];
this.data=new[bytes.size-4];
foreach(data[i])
data[i]= bytes[i +3];
this.fcs =0;
if(bytes[bytes.size-1]!= cal_fcs)
this.fcs =1;
endfunction: byte_unpack
```

14) Define a compare method.
Compares the current value of the object instance with the current value of the specified object instance.
If the value is different, FALSE is returned.

```
virtualfunctionbitcompare(packet pkt);
compare=1;
if(pkt ==null)
begin
$display(" ** ERROR ** : pkt : received a null object ");
compare=0;
end
else
begin
if(pkt.da !==this.da)
begin
$display(" ** ERROR **: pkt : Da field did not match");
```

```
compare=0;
end
if(pkt.sa !==this.sa)
begin
$display(" ** ERROR **: pkt : Sa field did not match");
compare=0;
end

if(pkt.length !==this.length)
begin
$display(" ** ERROR **: pkt : Length field did not match");
compare=0;
end
foreach(this.data[i])
if(pkt.data[i]!==this.data[i])
begin
$display(" ** ERROR **: pkt : Data[%0d] field did not match",i);
compare=0;
end

if(pkt.fcs !==this.fcs)
begin
$display(" ** ERROR **: pkt : fcs field did not match %h %h",pkt.fcs ,this.fcs);
compare=0;
end
end
endfunction:compare
```

Packet Class Source Code

```
`ifndefGUARD_PACKET
`defineGUARD_PACKET

//Define the enumerated types for packet types
typedefenum{GOOD_FCS,BAD_FCS} fcs_kind_t;
typedefenum{GOOD_LENGTH,BAD_LENGTH} length_kind_t;

class packet;
rand fcs_kind_t fcs_kind;
rand length_kind_t length_kind;

randbit[7:0] length;
randbit[7:0] da;
randbit[7:0] sa;
randbytedata[];//Payload using Dynamic array,size is generated on the fly
randbyte fcs;
```

```
constraint address_c { da inside{`P0,`P1,`P2,`P3};}

constraint payload_size_c {data.sizeinside{[1:255]};}

constraint length_kind_c {
(length_kind ==GOOD_LENGTH)-> length ==data.size;
(length_kind ==BAD_LENGTH)-> length ==data.size+2;}

constraint solve_size_length {solvedata.sizebefore length;}

constraint fcs_kind_c {
(fcs_kind ==GOOD_FCS)-> fcs ==8'b0;
(fcs_kind ==BAD_FCS)-> fcs ==8'b1;}

///// method to calculate the fcs /////
functionbyte cal_fcs;
integer i;
byte result ;
result =0;
result = result ^ da;
result = result ^ sa;
result = result ^ length;
for(i =0;i<data.size;i++)
result = result ^data[i];
result = fcs ^ result;
return result;
endfunction: cal_fcs

///// method to print the packet fields ////
virtualfunctionvoid display();
$display("\n---------------------- PACKET KIND ------------------------ ");
$display(" fcs_kind : %s ",fcs_kind.name());
$display(" length_kind : %s ",length_kind.name());
$display("-------- PACKET ---------- ");
$display(" 0 : %h ",da);
$display(" 1 : %h ",sa);
$display(" 2 : %h ",length);
foreach(data[i])
$write("%3d : %0h ",i +3,data[i]);
$display("\n %2d : %h ",data.size()+3, cal_fcs);
$display("------------------------------------------------------------ \n");
endfunction: display

///// method to pack the packet into bytes/////
virtualfunctionintunsigned byte_pack(reflogic[7:0] bytes[]);
bytes =new[data.size+4];
bytes[0]= da;
```

```
bytes[1]= sa;
bytes[2]= length;
foreach(data[i])
bytes[3+ i]=data[i];
bytes[data.size()+3]= cal_fcs;
byte_pack = bytes.size;
endfunction: byte_pack

////method to unpack the bytes in to packet /////
virtualfunctionvoid byte_unpack(constreflogic[7:0] bytes[]);
this.da = bytes[0];
this.sa = bytes[1];
this.length = bytes[2];
this.fcs = bytes[bytes.size-1];
this.data=new[bytes.size-4];
foreach(data[i])
data[i]= bytes[i +3];
this.fcs =0;
if(bytes[bytes.size-1]!= cal_fcs)
this.fcs =1;
endfunction: byte_unpack

//// method to compare the packets /////
virtualfunctionbitcompare(packet pkt);
compare=1;
if(pkt ==null)
begin
$display(" ** ERROR ** : pkt : received a null object ");
compare=0;
end
else
begin
if(pkt.da !==this.da)
begin
$display(" ** ERROR **: pkt : Da field did not match");
compare=0;
end
if(pkt.sa !==this.sa)
begin
$display(" ** ERROR **: pkt : Sa field did not match");
compare=0;
end
```

```
if(pkt.length !==this.length)
begin
$display(" ** ERROR **: pkt : Length field did not match");
compare=0;
end
foreach(this.data[i])
if(pkt.data[i]!==this.data[i])
begin
$display(" ** ERROR **: pkt : Data[%0d] field did not match",i);
compare=0;
end

if(pkt.fcs !==this.fcs)
begin
$display(" ** ERROR **: pkt : fcs field did not match %h %h",pkt.fcs ,this.fcs);
compare=0;
end
end
endfunction:compare

endclass
```

Now we will write a small program to test our packet implantation. This program block is not used to verify the DUT.

Write a simple program block and do the instance of packet class. Randomize the packet and call the display method to analyze the generation. Then pack the packet in to bytes and then unpack bytes and then call compare method to check all the methods.

Program Block Source Code

```
program test;

packet pkt1 =new();
packet pkt2 =new();
logic[7:0] bytes[];
initial
repeat(10)
if(pkt1.randomize)
begin
$display(" Randomization Successes full.");
pkt1.display();
void'(pkt1.byte_pack(bytes));
pkt2 =new();
pkt2.byte_unpack(bytes);
if(pkt2.compare(pkt1))
```

$display(" Packing,Unpacking and compare worked");
else
$display(" *** Something went wrong in Packing or Unpacking or compare ***");

end
else
$display(" *** Randomization Failed ***");

endprogram

(S)Download the packet class with program block.

switch_4.tar

(S)Run the simulation
vcs -sverilog -f filelist -R -ntb_opts dtm

(S)Log file report:

Randomization Sucessesfull.

---------- PACKET KIND ----------
fcs_kind : BAD_FCS
length_kind : GOOD_LENGTH
-------- PACKET ----------
0 : 00
1 : f7
2 : be
3 : a6 4 : 1b 5 : b5 6 : fa 7 : 4e 8 : 15 9 : 7d 10 : 72 11 : 96 12 : 31 13 : c4 14 : aa 15 : c4 16 : cf 17 : 4f 18 : f4 19 : 17 20 : 88 21 : f1 22 : 2c 23 : ce 24 : 5 25 : cb 26 : 8c 27 : 1a 28 : 37 29 : 60 30 : 5f 31 : 7a 32 : a2 33 : f0 34 : c9 35 : dc 36 : 41 37 : 3f 38 : 12 39 : f4 40 : df 41 : c5 42 : d7 43 : 94 44 : 88 45 : 1 46 : 31 47 : 29 48 : d6 49 : f4 50 : d9 51 : 4f 52 : 0 53 : dd 54 : d2 55 : a6 56 : 59 57 : 43 58 : 45 59 : f2 60 : a2 61 : a1 62 : fd 63 : ea 64 : c1 65 : 20 66 : c7 67 : 20 68 : e1 69 : 97 70 : c6 71 : cf 72 : cd 73 : 17 74 : 99 75 : 49 76 : b8 77 : 1c 78 : df 79 : e6 80 : 1a 81 : ce 82 : 8c 83 : ec 84 : b6 85 : bb 86 : a5 87 : 17 88 : cb 89 : 32 90 : e1 91 : 83 92 : 96 93 : e 94 : ee 95 : 57 96 : 33 97 : cd 98 : 62 99 : 88 100 : 7b 101 : e6 102 : 41 103 : ad 104 : 26 105 : ee 106 : 9c 107 : 95 108 : a7 109 : b8 110 : 83 111 : f 112 : ca 113 : ec 114 : b5 115 : 8d 116 : d8 117 : 2f 118 : 6f 119 : ea 120 : 4c 121 : 35 122 : 41 123 : f2 124 : 4e 125 : 89 126 : d8 127 : 78 128 : f1 129 : d 130 : d6 131 : d5 132 : 8 133 : c 134 : de 135 : a9 136 : 1d 137 : a0 138 : ae 139 : 99 140 : f5 141 : 53 142 : d8 143 : 7a 144 : 4c 145 : d4 146 : b8 147 : 54 148 : b7 149 : c3 150 : c9 151 : 7b 152 : a3 153 : 71 154 : 2b 155 : b4 156 : 50 157 : 54 158 : 22 159 : 95 160 : df 161 : 17 162 : c9 163 : 41 164 : 80 165 : 2b 166 : f0 167 : ba 168 : 4a 169 : a9 170 : 7f 171 : 13 172 : 1e 173 : 12 174 : a8 175 : 2 176 : 3 177 : 3d 178 : 71 179 : e6 180 : 96 181 : 89 182 : c6 183 : 46 184 : d6 185 : 1b 186 : 5f 187 : 20 188 : a0

189 : a3 190 : 49 191 : 79 192 : 9
193 : 53

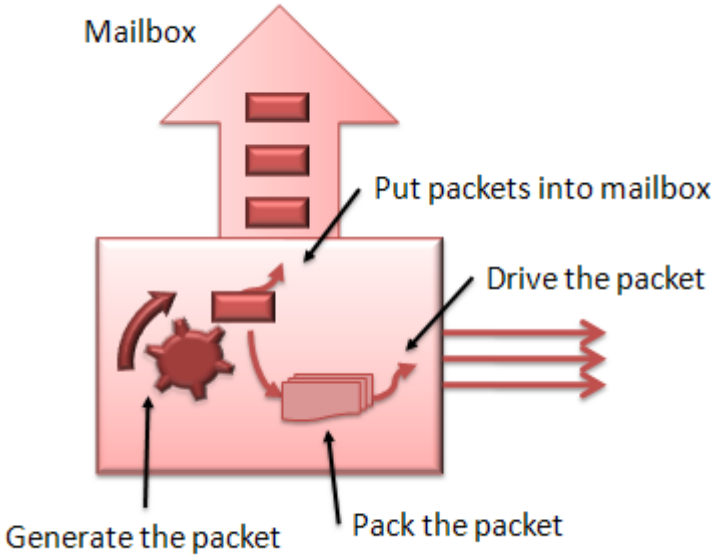
Packing,Unpacking and compare worked
Randomization Sucessesfull.

.....
.....
.....

PHASE 5 DRIVER

In phase 5 we will write a driver and then instantiate the driver in environment and send packet in to DUT. Driver class is defined in Driver.sv file.

Driver is class which generates the packets and then drives it to the DUT input interface and pushes the packet in to mailbox.



1) Declare a packet.

packet gpkt;

2) Declare a virtual input_interface of the switch. We will connect this to the Physical interface of the top module same as what we did in environment class.

virtual input_interface.IP input_intf;

3) Define a mailbox "drvr2sb" which is used to send the packets to the score board.

mailbox drvr2sb;

4) Define new constructor with arguments, virtual input interface and a mail box which is used to send packets from the driver to scoreboard.

functionnew(**virtual** input_interface.IP input_intf_new,mailbox drvr2sb);
this.input_intf = input_intf_new ;
if(drvr2sb ==**null**)

```
begin
$display(" **ERROR**: drvr2sb is null");
$finish;
end
else
this.drvr2sb = drvr2sb;
```

5) Construct the packet in the driver constructor.

```
gpkt =new();
```

6) Define the start method.
In start method, do the following

Repeat the following steps for num_of_pkts times.

```
repeat($root.num_of_pkts)
```

Randomize the packet and check if the randomization is successes full.

```
if( pkt.randomize())
begin
$display(" %0d : Driver : Randomization Successes full.",$time);
.....
.....
else
begin
$display(" %0d Driver : ** Randomization failed. **",$time);
.....
.....
```

Display the packet content.

```
pkt.display();
```

Then pack the packet in to bytes.

```
length = pkt.byte_pack(bytes);
```

Then send the packet byte in to the switch by asserting data_status of the input interface signal and driving the data bytes on to the data_in signal.

```
foreach(bytes[i])
begin
@(posedge input_intf.clock);
input_intf.cb.data_status <=1;
input_intf.cb.data_in <= bytes[i];
end
```

After driving all the data bytes, deassert data_status signal of the input interface.

```
@(posedge input_intf.clock);
input_intf.cb.data_status <=0;
input_intf.cb.data_in <=0;
```

Send the packet in to mail "drvr2sb" box for scoreboard.

```
drvr2sb.put(pkt);
```

If randomization fails, increment the error counter which is defined in Globals.sv file

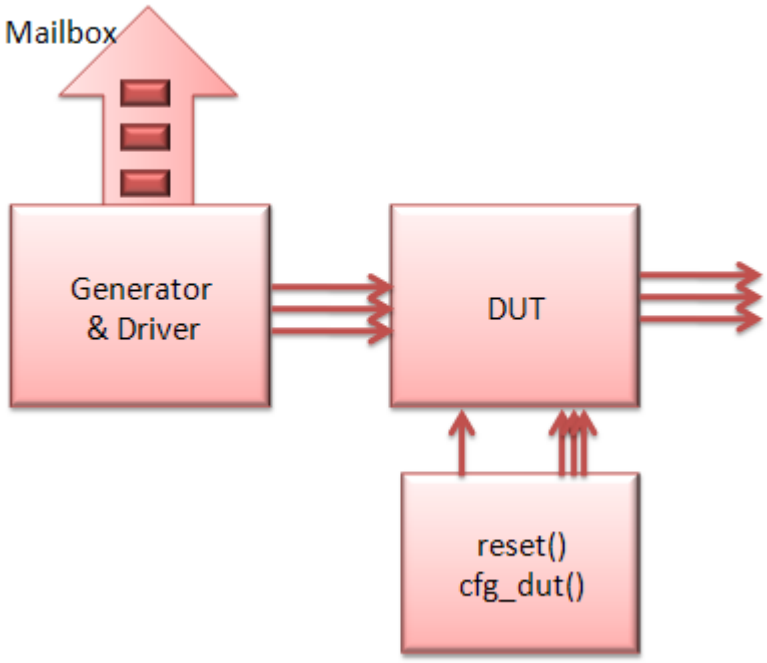
```
$root.error++;
```

Driver Class Source Code:

```
`ifndefGUARD_DRIVER
`defineGUARD_DRIVER

class Driver;
virtual input_interface.IP input_intf;
mailbox drvr2sb;
packet gpkt;

//// constructor method ////
functionnew(virtual input_interface.IP input_intf_new,mailbox drvr2sb);
this.input_intf = input_intf_new ;
if(drvr2sb ==null)
begin
$display(" **ERROR**: drvr2sb is null");
$finish;
end
else
this.drvr2sb = drvr2sb;
gpkt =new();
endfunction:new

/// method to send the packet to DUT ////////
taskstart();
packet pkt;
int length;
logic[7:0] bytes[];
repeat($root.num_of_pkts)
begin
repeat(3)@(posedge input_intf.clock);
pkt =new gpkt;
//// Randomize the packet /////
```

```
if( pkt.randomize())
begin
$display(" %0d : Driver : Randomization Successes full. ",$time);
//// display the packet content ///////
pkt.display();

//// Pack the packet in tp stream of bytes //////
length = pkt.byte_pack(bytes);

///// assert the data_status signal and send the packed bytes //////
foreach(bytes[i])
begin
@(posedge input_intf.clock);
input_intf.cb.data_status <=1;
input_intf.cb.data_in <= bytes[i];
end

//// deassert the data_status singal //////
@(posedge input_intf.clock);
input_intf.cb.data_status <=0;
input_intf.cb.data_in <=0;

//// Push the packet in to mailbox for scoreboard /////
drvr2sb.put(pkt);

$display(" %0d : Driver : Finished Driving the packet with length %0d",$time,length);
end
else
begin
$display(" %0d Driver : ** Randomization failed. **",$time);
////// Increment the error count in randomization fails ////////
$root.error++;
end
end
endtask:start

endclass

`endif
```

Now we will take the instance of the driver in the environment class.



1) Declare a mailbox "drvr2sb" which will be used to connect the scoreboard and driver.

mailbox drvr2sb;

2) Declare a driver object "drvr".

Driver drvr;

3) In build method, construct the mail box.

drvr2sb =new();

4) In build method, construct the driver object. Pass the input_intf and "drvr2sb" mail box.

drvr=new(input_intf,drvr2sb);

5) To start sending the packets to the DUT, call the start method of "drvr" in the start method of Environment class.

drvr.start();

Environment Class Source Code:

```
`ifndef GUARD_ENV
`define GUARD_ENV

class Environment ;
```

```
virtual mem_interface.MEM mem_intf ;
virtual input_interface.IP input_intf ;
virtual output_interface.OP output_intf[4] ;

Driver drvr;
mailbox drvr2sb;

function new(virtual mem_interface.MEM mem_intf_new ,
virtual input_interface.IP input_intf_new ,
virtual output_interface.OP output_intf_new[4] );

this.mem_intf = mem_intf_new ;
this.input_intf = input_intf_new ;
this.output_intf = output_intf_new ;

$display(" %0d : Environment : created env object",$time);
endfunction : new

function void build();
$display(" %0d : Environment : start of build() method",$time);

drvr2sb =new();
drvr=new(input_intf,drvr2sb);

$display(" %0d : Environment : end of build() method",$time);
endfunction : build

task reset();
$display(" %0d : Environment : start of reset() method",$time);
// Drive all DUT inputs to a known state
mem_intf.cb.mem_data <= 0;
mem_intf.cb.mem_add <= 0;
mem_intf.cb.mem_en <= 0;
mem_intf.cb.mem_rd_wr <= 0;
input_intf.cb.data_in <= 0;
input_intf.cb.data_status <= 0;
output_intf[0].cb.read <= 0;
output_intf[1].cb.read <= 0;
output_intf[2].cb.read <= 0;
output_intf[3].cb.read <= 0;

// Reset the DUT
input_intf.reset <= 1;
repeat (4) @ input_intf.clock;
input_intf.reset <= 0;
```

```
$display(" %0d : Environment : end of reset() method",$time);
endtask : reset

task cfg_dut();
$display(" %0d : Environment : start of cfg_dut() method",$time);

mem_intf.cb.mem_en <= 1;
@(posedge mem_intf.clock);
mem_intf.cb.mem_rd_wr <= 1;

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h0;
mem_intf.cb.mem_data <= `P0;
$display(" %0d : Environment : Port 0 Address %h ",$time,`P0);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h1;
mem_intf.cb.mem_data <= `P1;
$display(" %0d : Environment : Port 1 Address %h ",$time,`P1);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h2;
mem_intf.cb.mem_data <= `P2;
$display(" %0d : Environment : Port 2 Address %h ",$time,`P2);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h3;
mem_intf.cb.mem_data <= `P3;
$display(" %0d : Environment : Port 3 Address %h ",$time,`P3);

@(posedge mem_intf.clock);
mem_intf.cb.mem_en <=0;
mem_intf.cb.mem_rd_wr <= 0;
mem_intf.cb.mem_add <= 0;
mem_intf.cb.mem_data <= 0;


$display(" %0d : Environment : end of cfg_dut() method",$time);
endtask :cfg_dut

task start();
$display(" %0d : Environment : start of start() method",$time);

drvr.start();
```

```
$display(" %0d : Environment : end of start() method",$time);
endtask : start

task wait_for_end();
$display(" %0d : Environment : start of wait_for_end() method",$time);
repeat(10000) @(input_intf.clock);
$display(" %0d : Environment : end of wait_for_end() method",$time);
endtask : wait_for_end

task run();
$display(" %0d : Environment : start of run() method",$time);
build();
reset();
cfg_dut();
start();
wait_for_end();
report();
$display(" %0d : Environment : end of run() method",$time);
endtask : run

task report();
endtask : report
endclass

`endif
```

(S)Download the phase 5 source code:

switch_5.tar

(S)Run the command:

vcs -sverilog -f filelist -R -ntb_opts dtm

(S)Log file report.

```
****************** Start of testcase ****************
0 : Environment : created env object
0 : Environment : start of run() method
0 : Environment : start of build() method
0 : Environment : end of build() method
0 : Environment : start of reset() method
40 : Environment : end of reset() method
40 : Environment : start of cfg_dut() method
70 : Environment : Port 0 Address 00
90 : Environment : Port 1 Address 11
110 : Environment : Port 2 Address 22
130 : Environment : Port 3 Address 33
```

150 : Environment : end of cfg_dut() method
150 : Environment : start of start() method
210 : Driver : Randomization Successes full.

---------- PACKET KIND ----------
fcs_kind : BAD_FCS
length_kind : GOOD_LENGTH
-------- PACKET ----------
0 : 22
1 : 11
2 : 2d
3 : 63 4 : 2a 5 : 2e 6 : c 7 : a 8 : 14 9 : c1 10 : 14 11 : 8f 12 : 54 13 : 5d 14 : da 15 : 22 16 : 2c 17 : ac 18 : 1c 19 : 48 20 : 3c 21 : 7e 22 : f3 23 : ed 24 : 24 25 : d1 26 : 3e 27 : 38 28 : aa 29 : 54 30 : 19 31 : 89 32 : aa 33 : cf 34 : 67 35 : 19 36 : 9a 37 : 1d 38 : 96 39 : 8 40 : 15 41 : 66 42 : 55 43 : b 44 : 70 45 : 35 46 : fc 47 : 8f
48 : cd

1210 : Driver : Finished Driving the packet with length 49
1270 : Driver : Randomization Successes full.

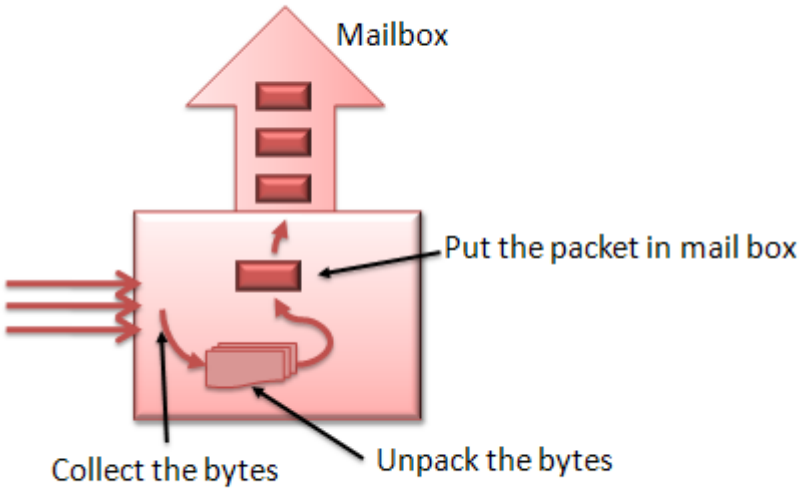
.....
.....
.....

PHASE 6 RECEIVER

In this phase, we will write a receiver and use the receiver in environment class to collect the packets coming from the switch output_interface.

Receiver collects the data bytes from the interface signal. And then unpacks the bytes in to packet and pushes it into mailbox.

Receiver class is written in reveicer.sv file.



1) Declare a virtual output_interface. We will connect this to the Physical interface of the top module, same as what we did in environment class.

virtual output_interface.OP output_intf;

2) Declare a mailbox "rcvr2sb" which is used to send the packets to the score board

mailbox rcvr2sb;

3) Define new constructor with arguments, virtual input interface and a mail box which is used to send packets from the receiver to scoreboard.

```
functionnew(virtual output_interface.OP output_intf_new,mailbox rcvr2sb);
this.output_intf = output_intf_new ;
if(rcvr2sb ==null)
begin
$display(" **ERROR**: rcvr2sb is null");
```

```
$finish;
end
else
this.rcvr2sb = rcvr2sb;
endfunction:new
```

4) Define the start method.

In start method, do the following

Wait for the ready signal to be asserted by the DUT.

```
wait(output_intf.cb.ready)
```

If the ready signal is asserted, then request the DUT to send the data out from the data_out signal by asserting the read signal. When the data to be sent is finished by the DUT, it will deassert the ready signal. Once the ready signal is deasserted, stop collecting the data bytes and deasseart the read signal.

```
output_intf.cb.read <=1;
repeat(2)@(posedge output_intf.clock);
while(output_intf.cb.ready)
begin
bytes =new[bytes.size+1](bytes);
bytes[bytes.size-1]= output_intf.cb.data_out;
@(posedge output_intf.clock);
end
output_intf.cb.read <=0;
@(posedge output_intf.clock);
$display(" %0d : Receiver : Received a packet of length %0d",$time,bytes.size);
```

Create a new packet object of packet.

```
pkt =new();
```

Then call the unpack method of the packet to unpacked the bytes and then display the packet content.

```
pkt.byte_unpack(bytes);
pkt.display();
```

Then send the packet to scoreboard.

```
rcvr2sb.put(pkt);
```

Delete the dynamic array bytes.

```
bytes.delete();
```

Receiver Class Source Code:

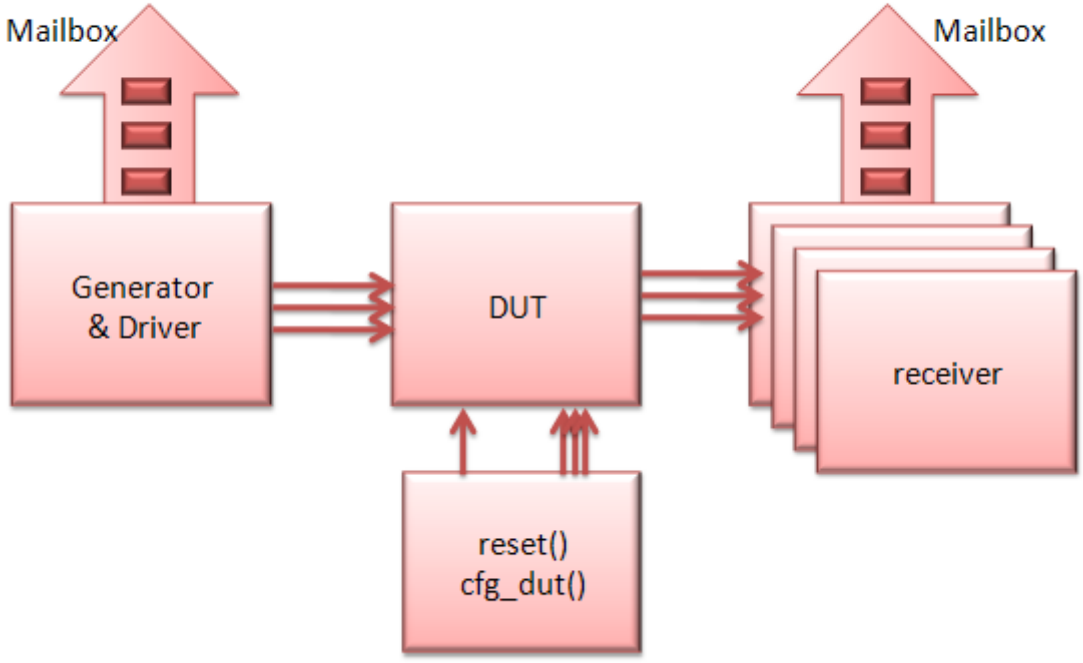
```
`ifndefGUARD_RECEIVER
`defineGUARD_RECEIVER

class Receiver;

virtual output_interface.OP output_intf;
mailbox rcvr2sb;

//// constructor method ////
functionnew(virtual output_interface.OP output_intf_new,mailbox rcvr2sb);
this.output_intf = output_intf_new ;
if(rcvr2sb ==null)
begin
$display(" **ERROR**: rcvr2sb is null");
$finish;
end
else
this.rcvr2sb = rcvr2sb;
endfunction:new

taskstart();
logic[7:0] bytes[];
packet pkt;
forever
begin
repeat(2)@(posedge output_intf.clock);
wait(output_intf.cb.ready)
output_intf.cb.read <=1;
repeat(2)@(posedge output_intf.clock);
while(output_intf.cb.ready)
begin
bytes =new[bytes.size+1](bytes);
bytes[bytes.size-1]= output_intf.cb.data_out;
@(posedge output_intf.clock);
end
output_intf.cb.read <=0;
@(posedge output_intf.clock);
$display(" %0d : Receiver : Received a packet of length %0d",$time,bytes.size);
pkt =new();
pkt.byte_unpack(bytes);
pkt.display();
rcvr2sb.put(pkt);
```

bytes.delete();
end
endtask:start

endclass

`endif

Now we will take the instance of the receiver in the environment class.



1) Declare a mailbox "rcvr2sb" which will be used to connect the scoreboard and receiver.

mailbox rcvr2sb;

2) Declare 4 receiver object "rcvr".

Receiver rcvr[4];

3) In build method, construct the mail box.

rcvr2sb =new();

4) In build method, construct the receiver object. Pass the output_intf and "rcvr2sb" mail box. There are 4 output interfaces and receiver objects. We will connect one receiver for one output interface.

foreach(rcvr[i])
rcvr[i]=new(output_intf[i],rcvr2sb);

5) To start collecting the packets from the DUT, call the "start" method of "rcvr" in the "start" method of Environment class.

```
taskstart();
$display(" %0d : Environment : start of start() method",$time);
fork
drvr.start();
rcvr[0].start();
rcvr[1].start();
rcvr[2].start();
rcvr[3].start();
join_any
$display(" %0d : Environment : end of start() method",$time);
endtask:start
```

Environment Class Source Code:

```
`ifndef GUARD_ENV
`define GUARD_ENV

class Environment ;

virtual mem_interface.MEM mem_intf ;
virtual input_interface.IP input_intf ;
virtual output_interface.OP output_intf[4] ;


Driver drvr;

Receiver rcvr[4];

mailbox drvr2sb;

mailbox rcvr2sb;

function new(virtual mem_interface.MEM mem_intf_new ,
virtual input_interface.IP input_intf_new ,
virtual output_interface.OP output_intf_new[4] );

this.mem_intf = mem_intf_new ;
this.input_intf = input_intf_new ;
this.output_intf = output_intf_new ;


$display(" %0d : Environment : created env object",$time);
endfunction : new

function void build();
$display(" %0d : Environment : start of build() method",$time);
drvr2sb = new();
```

```
rcvr2sb =new();

drvr= new(input_intf,drvr2sb);

foreach(rcvr[i])
rcvr[i]=new(output_intf[i],rcvr2sb);

$display(" %0d : Environment : end of build() method",$time);
endfunction : build

task reset();
$display(" %0d : Environment : start of reset() method",$time);
// Drive all DUT inputs to a known state
mem_intf.cb.mem_data <= 0;
mem_intf.cb.mem_add <= 0;
mem_intf.cb.mem_en <= 0;
mem_intf.cb.mem_rd_wr <= 0;
input_intf.cb.data_in <= 0;
input_intf.cb.data_status <= 0;
output_intf[0].cb.read <= 0;
output_intf[1].cb.read <= 0;
output_intf[2].cb.read <= 0;
output_intf[3].cb.read <= 0;

// Reset the DUT
input_intf.reset <= 1;
repeat (4) @ input_intf.clock;
input_intf.reset <= 0;

$display(" %0d : Environment : end of reset() method",$time);
endtask : reset

task cfg_dut();
$display(" %0d : Environment : start of cfg_dut() method",$time);

mem_intf.cb.mem_en <= 1;
@(posedge mem_intf.clock);
mem_intf.cb.mem_rd_wr <= 1;

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h0;
mem_intf.cb.mem_data <= `P0;
$display(" %0d : Environment : Port 0 Address %h ",$time,`P0);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h1;
```

```
mem_intf.cb.mem_data <= `P1;
$display(" %0d : Environment : Port 1 Address %h ",$time,`P1);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h2;
mem_intf.cb.mem_data <= `P2;
$display(" %0d : Environment : Port 2 Address %h ",$time,`P2);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h3;
mem_intf.cb.mem_data <= `P3;
$display(" %0d : Environment : Port 3 Address %h ",$time,`P3);

@(posedge mem_intf.clock);
mem_intf.cb.mem_en <=0;
mem_intf.cb.mem_rd_wr <= 0;
mem_intf.cb.mem_add <= 0;
mem_intf.cb.mem_data <= 0;


$display(" %0d : Environment : end of cfg_dut() method",$time);
endtask :cfg_dut

task start();
$display(" %0d : Environment : start of start() method",$time);
fork
drvr.start();

rcvr[0].start();
rcvr[1].start();
rcvr[2].start();
rcvr[3].start();

join_any
$display(" %0d : Environment : end of start() method",$time);
endtask : start

task wait_for_end();
$display(" %0d : Environment : start of wait_for_end() method",$time);
repeat(10000) @(input_intf.clock);
$display(" %0d : Environment : end of wait_for_end() method",$time);
endtask : wait_for_end

task run();
$display(" %0d : Environment : start of run() method",$time);
build();
```

```
reset();
cfg_dut();
start();
wait_for_end();
report();
$display(" %0d : Environment : end of run() method",$time);

endtask : run

task report();
endtask: report
endclass

`endif
```

(S)Download the phase 6 source code:

switch_6.tar

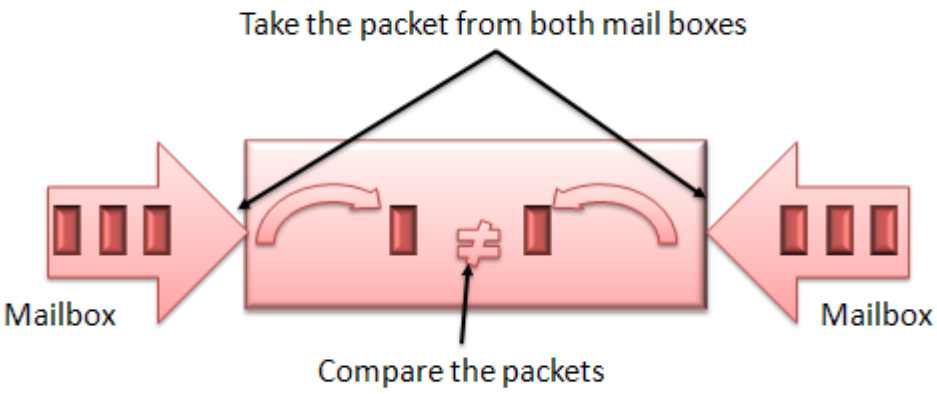
(S)Run the command:

```
vcs -sverilog -f filelist -R -ntb_opts dtm
```

PHASE 7 SCOREBOARD

In this phase we will see the scoreboard implementation.

Scoreboard has 2 mailboxes. One is used to for getting the packets from the driver and other from the receiver. Then the packets are compared and if they don't match, then error is asserted.
Scoreboard in implemented in file Scoreboard.sv.



1) Declare 2 mailboxes drvr2sb and rcvr2sb.

```
mailbox drvr2sb;
mailbox rcvr2sb;
```

2) Declare a constructor method with "drvr2sb" and "rcvr2sb" mailboxes as arguments.

```
function new(mailbox drvr2sb,mailbox rcvr2sb);
```

3) Connect the mailboxes of the constructor to the mail boxes of the scoreboard.

```
this.drvr2sb = drvr2sb;
this.rcvr2sb = rcvr2sb;
```

4) Define a start method.
Do the following steps forever.
Wait until there is a packet is in "rcvr2sb". Then pop the packet from the mail box.

```
rcvr2sb.get(pkt_rcv);
$display(" %0d : Scorebooard : Scoreboard received a packet from receiver ",$time);
```

Then pop the packet from drvr2sb.

```
drvr2sb.get(pkt_exp);
```

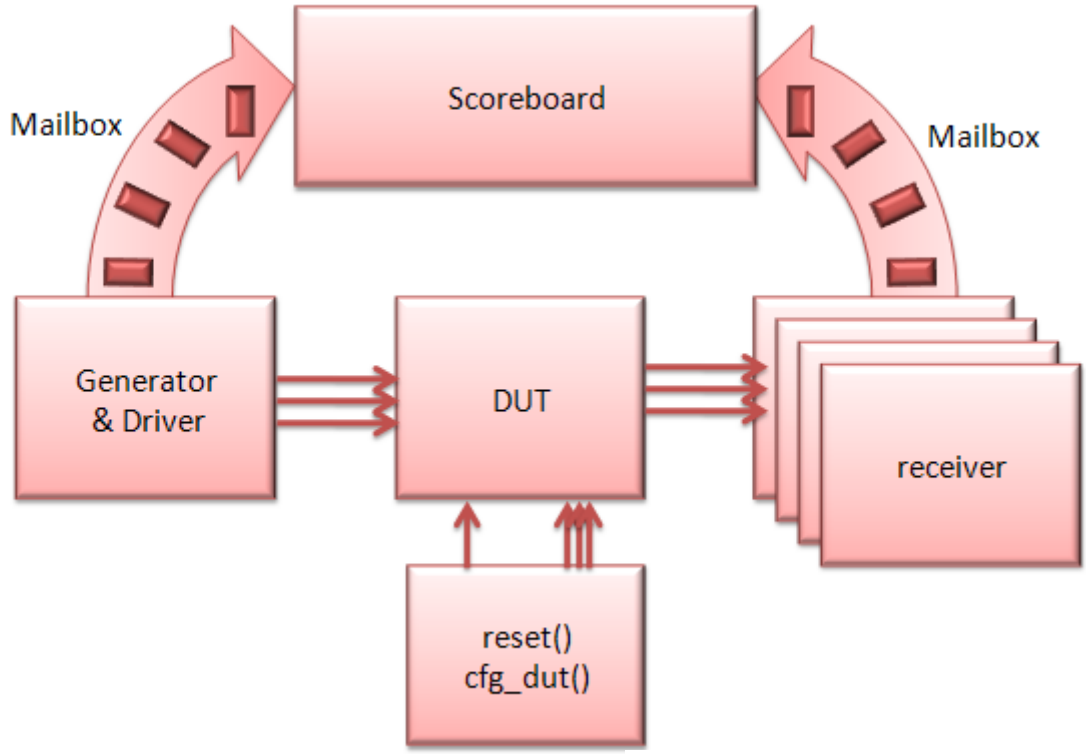
Compare both packets and increment an error counter if they are not equal.

```
if(pkt_rcv.compare(pkt_exp))
$display(" %0d : Scoreboardd :Packet Matched ",$time);
else
$root.error++;
```

Scoreboard Class Source Code:

```
`ifndefGUARD_SCOREBOARD
`defineGUARD_SCOREBOARD

class Scoreboard;

mailbox drvr2sb;
mailbox rcvr2sb;

functionnew(mailbox drvr2sb,mailbox rcvr2sb);
this.drvr2sb = drvr2sb;
this.rcvr2sb = rcvr2sb;
endfunction:new

taskstart();
packet pkt_rcv,pkt_exp;
forever
begin
rcvr2sb.get(pkt_rcv);
$display(" %0d : Scorebooard : Scoreboard received a packet from receiver ",$time);
drvr2sb.get(pkt_exp);
if(pkt_rcv.compare(pkt_exp))
$display(" %0d : Scoreboardd :Packet Matched ",$time);
else
$root.error++;
end
endtask:start

endclass

`endif
```

Now we will see how to connect the scoreboard in the Environment class.



1) Declare a scoreboard.

Scoreboard sb;

2) Construct the scoreboard in the build method. Pass the drvr2sb and rcvr2sb mailboxes to the score board constructor.

sb =new(drvr2sb,rcvr2sb);

3) Start the scoreboard method in the start method.

sb.start();

4) Now we are to the end of building the verification environment.
In the report() method of environment class, print the TEST PASS or TEST FAIL status based on the error count.

```
task report();
$display("\n\n*************************************************");
if(0==$root.error)
$display("******** TEST PASSED *********");
else
$display("******** TEST Failed with %0d errors *********",$root.error);

$display("*************************************************\n\n");
endtask: report
```

Source Code Of The Environment Class:

```
`ifndef GUARD_ENV
`define GUARD_ENV

class Environment ;

virtual mem_interface.MEM mem_intf ;
virtual input_interface.IP input_intf ;
virtual output_interface.OP output_intf[4] ;


Driver drvr;
Receiver rcvr[4];

Scoreboard sb;

mailbox drvr2sb ;
mailbox rcvr2sb ;

function new(virtual mem_interface.MEM mem_intf_new ,
virtual input_interface.IP input_intf_new ,
virtual output_interface.OP output_intf_new[4] );

this.mem_intf = mem_intf_new ;
this.input_intf = input_intf_new ;
this.output_intf = output_intf_new ;

$display(" %0d : Environment : created env object",$time);
endfunction : new

function void build();
$display(" %0d : Environment : start of build() method",$time);
drvr2sb = new();
rcvr2sb = new();

sb =new(drvr2sb,rcvr2sb);

drvr= new(input_intf,drvr2sb);
foreach(rcvr[i])
rcvr[i]= new(output_intf[i],rcvr2sb);
$display(" %0d : Environment : end of build() method",$time);
endfunction : build

task reset();
$display(" %0d : Environment : start of reset() method",$time);
// Drive all DUT inputs to a known state
mem_intf.cb.mem_data <= 0;
mem_intf.cb.mem_add <= 0;
```

```
mem_intf.cb.mem_en <= 0;
mem_intf.cb.mem_rd_wr <= 0;
input_intf.cb.data_in <= 0;
input_intf.cb.data_status <= 0;
output_intf[0].cb.read <= 0;
output_intf[1].cb.read <= 0;
output_intf[2].cb.read <= 0;
output_intf[3].cb.read <= 0;

// Reset the DUT
input_intf.reset <= 1;
repeat (4) @ input_intf.clock;
input_intf.reset <= 0;

$display(" %0d : Environment : end of reset() method",$time);
endtask : reset

task cfg_dut();
$display(" %0d : Environment : start of cfg_dut() method",$time);

mem_intf.cb.mem_en <= 1;
@(posedge mem_intf.clock);
mem_intf.cb.mem_rd_wr <= 1;

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h0;
mem_intf.cb.mem_data <= `P0;
$display(" %0d : Environment : Port 0 Address %h ",$time,`P0);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h1;
mem_intf.cb.mem_data <= `P1;
$display(" %0d : Environment : Port 1 Address %h ",$time,`P1);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h2;
mem_intf.cb.mem_data <= `P2;
$display(" %0d : Environment : Port 2 Address %h ",$time,`P2);

@(posedge mem_intf.clock);
mem_intf.cb.mem_add <= 8'h3;
mem_intf.cb.mem_data <= `P3;
$display(" %0d : Environment : Port 3 Address %h ",$time,`P3);

@(posedge mem_intf.clock);
```

```
mem_intf.cb.mem_en <=0;
mem_intf.cb.mem_rd_wr <= 0;
mem_intf.cb.mem_add <= 0;
mem_intf.cb.mem_data <= 0;


$display(" %0d : Environment : end of cfg_dut() method",$time);
endtask :cfg_dut

task start();
$display(" %0d : Environment : start of start() method",$time);
fork
drvr.start();
rcvr[0].start();
rcvr[1].start();
rcvr[2].start();
rcvr[3].start();

sb.start();

join_any
$display(" %0d : Environment : end of start() method",$time);
endtask : start

task wait_for_end();
$display(" %0d : Environment : start of wait_for_end() method",$time);
repeat(10000) @(input_intf.clock);
$display(" %0d : Environment : end of wait_for_end() method",$time);
endtask : wait_for_end

task run();
$display(" %0d : Environment : start of run() method",$time);
build();
reset();
cfg_dut();
start();
wait_for_end();
report();
$display(" %0d : Environment : end of run() method",$time);
endtask: run

task report();

$display("\n\n*********************************************");
if(0==$root.error)
$display("******** TEST PASSED *********");
```

else
$display("******** TEST Failed with %0d errors *********",$root.error);

$display("**\n\n");

endtask : report

endclass
`endif

(S)Download the phase 7 score code:

switch_7.tar

(S)Run the simulation:
vcs -sverilog -f filelist -R -ntb_opts dtm

PHASE 8 COVERAGE

In this phase we will write the functional coverage for switch protocol. Functional coverage is written in Coverage.sv file. After running simulation, you will analyze the coverage results and find out if some test scenarios have not been exercised and write tests to exercise them.

The points which we need to cover are
1) Cover all the port address configurations.
2) Cover all the packet lengths.
3) Cover all correct and incorrect length fields.
4) Cover good and bad FCS.
5) Cover all the above combinations.

1) Define a cover group with following cover points.

a) All packet lengths:

length :**coverpoint** pkt.length;

b) All port address:

da :**coverpoint** pkt.da {
bins p0 ={`P0};
bins p1 ={`P1};
bins p2 ={`P2};
bins p3 ={`P3};}

c) Correct and incorrect Length field types:

length_kind :**coverpoint** pkt.length_kind;

d) Good and Bad FCS:

fcs_kind :**coverpoint** pkt.fcs_kind;

5) Cross product of all the above cover points:

all_cross:**cross** length,da,length_kind,fcs_kind;

2) In constructor method, construct the cover group

functionnew();
switch_coverage =new();
endfunction:new

3) Write task which calls the sample method to cover the points.

task sample(packet pkt);
this.pkt = pkt;
switch_coverage.sample();
endtask:sample

Source Code Of Coverage Class:

`ifndefGUARD_COVERAGE
`defineGUARD_COVERAGE

class coverage;
packet pkt;

covergroup switch_coverage;

length :**coverpoint** pkt.length;
da :**coverpoint** pkt.da {
bins p0 ={`P0};
bins p1 ={`P1};
bins p2 ={`P2};
bins p3 ={`P3};}
length_kind :**coverpoint** pkt.length_kind;
fcs_kind :**coverpoint** pkt.fcs_kind;

all_cross:**cross** length,da,length_kind,fcs_kind;
endgroup

functionnew();
switch_coverage =new();
endfunction:new

task sample(packet pkt);
this.pkt = pkt;
switch_coverage.sample();
endtask:sample

endclass

`endif

Now we will use this coverage class instance in scoreboard.

1) Take an instance of coverage class and construct it in scoreboard class.

coverage cov =new();

2) Call the sample method and pass the exp_pkt to the sample method.

cov.sample(pkt_exp);

Source Code Of The Scoreboard Class:

```
`ifndef GUARD_SCOREBOARD
`define GUARD_SCOREBOARD

class Scoreboard;

mailbox drvr2sb;
mailbox rcvr2sb;

coverage cov =new();

function new(mailbox drvr2sb,mailbox rcvr2sb);
this.drvr2sb = drvr2sb;
this.rcvr2sb = rcvr2sb;
endfunction:new

task start();
packet pkt_rcv,pkt_exp;
forever
begin
rcvr2sb.get(pkt_rcv);
$display(" %0d : Scorebooard : Scoreboard received a packet from receiver ",$time);
drvr2sb.get(pkt_exp);
if(pkt_rcv.compare(pkt_exp))
begin
$display(" %0d : Scoreboardd :Packet Matched ",$time);

cov.sample(pkt_exp);

end
else
$root.error++;
end
endtask : start

endclass

`endif
```

(S)Download the phase 8 score code:

switch_8.tar

(S)Run the simulation:

vcs -sverilog -f filelist -R -ntb_opts dtm
urg -dir simv.cm

PHASE 9 TESTCASE

In this phase we will write a constraint random testcase.
Lets verify the DUT by sending large packets of length above 200.

1) In testcase file, define a small_packet class.
This calls is inherited from the packet class and data.size() field is constraint to generate the packet with size greater than 200.

class small_packet **extends** packet;

constraint small_c {data.size>200;}

endclass

2) In program block, create an object of the small_packet class.
Then call the build method of env.

small_packet spkt;

3) Pass the object of the small_packet to the packet handle which is in driver.

env.drvr.gpkt = spkt;

Then call the reset(),cfg_dut(),start(),wait_for_end() and report() methods as in the run method.

env.reset();
env.cfg_dut();
env.start();
env.wait_for_end();
env.report();

Source Code Of Constraint Testcase:

`ifndefGUARD_TESTCASE
`defineGUARD_TESTCASE

class small_packet **extends** packet;

constraint small_c {data.size>200;}

endclass

```
program testcase(mem_interface.MEM mem_intf,input_interface.IP input_intf,output_interface.OP output_intf[4]);

Environment env;
small_packet spkt;

initial
begin
$display(" ******************* Start of testcase *****************");
spkt =new();
env =new(mem_intf,input_intf,output_intf);
env.build();
env.drvr.gpkt = spkt;
env.reset();
env.cfg_dut();
env.start();
env.wait_for_end();
env.report();
#1000;
end

final
$display(" ******************** End of testcase ******************");

endprogram
`endif
```

(S)Download the phase 9 source code:

switch_9.tar

(S)Run the simulation:

```
vcs -sverilog -f filelist -R -ntb_opts dtm
urg -dir simv.cm
```