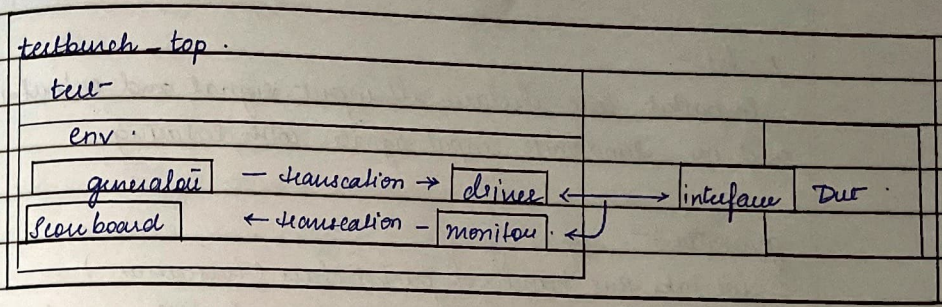
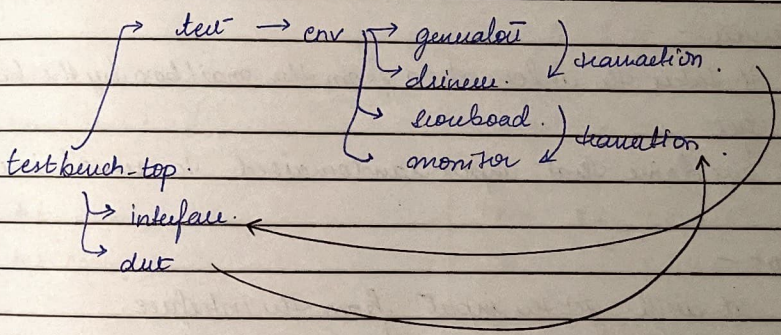
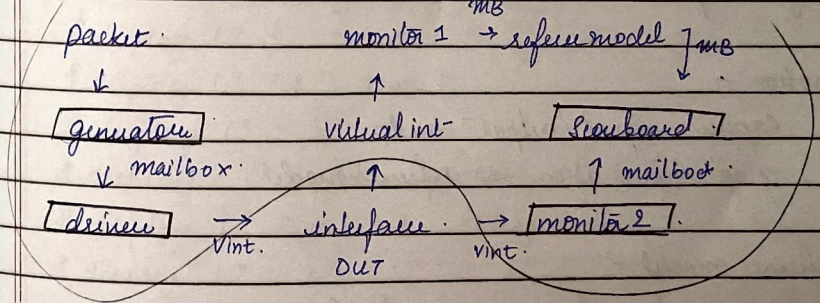
layer testbench. 18th Nov 2028.



test:- generate the stimulus this will go through interface and go to dut

dut:- dut will give o/p which we are going to capture and check with the expected value. which will tell pass or fail.





Lets assume for HA.

packet:-
In packet we declare all input signal and output signal and we randomise input signals. with declaring

generator:-
1. we take this handle of packet class (transcation).
2. we randomise the packet input signal with rand mode.
3. send the randomise input to the mailbox.

why mailbox ? :- it has two methods put and get.

driver:-
1. it takes the information from the mailbox by the help of get
2. we drive that input randomaised to virtual interface.

dut:-
1. it will get the input from the interface.
2. you capature the information by the dut in monitor

motion:-
1. capture input/output and.
2. gives information to refernce model.

refernce model:-
1. we create a logic similar to design
2. you will have 2 output; one from DUT one from refernce model.

Scoreboard:-

1. If the output from both drivers are equal then pass or b/s fail
2. Sometimes reference model are inside scoreboard itself.

Why we need second monitor (not in portion of project)

1 monitor 2 has output whereas monitor 1 has input. because reference model needs only input

coding :- (code is written for HA so that you can understand better with simple design. in notes it is compared to our project).

packet → transaction :-

class transaction.

rand bit a;
rand bit b;
} randomising

bit sum;
bit carry;

imp

design.sv :-

module half adder (s, c, a, b);
input a, b;
output s, c;

xor x1 (s, a, b);
and a1 (c, a, b);
endmodule

function void display (string name);
$display ("-----------");
$display ("%s", name);
$display ("-----------");
$display ("a = %d, b = %d", a, b);
$display ("sum = %d, carray = %d, sum, carry);
$display ("-----------")
endfunction.
endclass

*Note:- in our project layer 1b you can see it difined as fib_trans. and we have randomise the only input i and displayed

generator :-

class generator; handle of packet (transaction).
↑
transaction trans; property No. 1
mailbox gen2driv; property No. 3.
↳ mailbox is a class whose have declared here.

function new (mailbox gen2driv); created a construction of the mailbox gen2driv
this.gen2driv = gen2driv;
↳ basically telling whenever the mailbox is called with gen2driv that will be nothing but the mailbox gen2driv in class generator.

task main():
why?
the whole task will be repeated once
repeat(1)
begin.
trans = new() → calling the packet class.
↑ basically creating obj trans.
trans.randomize(); property No. 2
trans.display("Generator"); it is the string which is used to display.
← calling display function.
gen2driv.put(trans);
↳ putting the output which is basically the randomise input of design in the mailbox to send from generator to driver.
end
end task
end class.

classmate
Date
Page

*Note:-
in our project instead of telling repeat(1) we made a function to construction based on mail box but also give a count when to end the event.
We are also checking if the input are randmized or not to make sure there is no error.

rand fib_trans trans:-

end :-
→ endid;

interface:-
1. basically your interface is static and rest are dynamic
2. it is not possible to communicate directly from dynamic to static hence virtual interface.
3. basically for class when we do new(); then the memory is allocated
4. eg (DRIVER) Class $\xrightarrow{VI}$ interface.

virtual interface:- is a pointer of interface.

driver:-

class driver;

→ given name of your interface
keyword — virtual intf vif; → handle of virtual interface
mailbox gen2driv; → declaration.

PTO.

classmate
Date
Page

↳ created a constructor include & variables

function new (virtual intf vif, mailbox gen2driv);
this.vif = vif; ↳ pointer to interface.
this.gen2driv = gen2driv
endfunction.

task main;

repeat(1)
begin
transaction trans; calling object pkt.
gen2driv.get(trans); property 1
↳ used only for input of design.
vif.a <= trans.a;
what is <= ?
↳ non blocking assignment - so in order they are calculated at the last.
vif.b <= trans.b;
trans.sum = vif.sum;
↳ used for output of design
not imp. { trans.carry = vif.carry;
trans.display ("Driver");
↳ calling the display function of class trans.

end
endtask.

why do we display again and again?

endclass.

* Note :- in our project the design is fib_bfm.

1. because our design is sequential we have to consider reset.

task reset;
wait (intf.rst); → we are waiting until we receive a rst signal, this is basically like sensitive list.
$display ("Reset Initiated");
intf.bfm_cb.start <= 0; } initialing start FSM and
intf.bfm_cb.i <= 0; } input i to 0
wait (!intf.rst); → waiting until not rst.
$display ("Reset finished");
endtask:

2. we are also displaying the no. of transaction and incrementing it with every transaction.

3. we have repeated (2) @ (posedge intf.clk) because.

interface :-

interface inf(); → notice this isn't a class
  logic a;
  logic b;
  logic sum;
  logic carry;
endinterface

* Note :- usually interface contains 2 thing one is clockblock and modport this HA design is small hence its not required here.

* Note:-

in our project we do have clocking block and modport

1. one thing to remember is that you can't randomise your input here. Sobasically under clocking bfm cb @ (clock posedge clk) your input will convert into output and vice-versa. it is directing them at what clock they would act

2. there are no output in monitor so all your logic under clocking monitor cb @ (posedge clk) will be inputs

3. what do you mean by default #1 input output #1

4. here we have instated modport BFM and modport MONITOR. with the same logic for monitor and BFM logic significance.

* Note:-

the clock blocking and modport are inbuilt function of the system verilog hence there is no separate module for that.

monitor;

```
class monitor;
    virtual intf vif;          } declaration
    mailbox mon2scb            }

    function new (virtual intf vif, mailbox mon2scb);
    this.vif = vif;            }
    this.mon2scb = mon2scb;    } constructor.
    end function

    task main;
        repeat (1)
        #3;
        begin.                 → handle for the packet
        transaction trans;
        trans = new();         → object for the packet
        trans.a = vif.a;       }
        trans.b = vif.b;       }
        trans.sum = vif.sum;   } It is taking the information
        trans.carry = vif.carry; } from virtual interface
        mon2scb.put(trans);    } and putting it in a
        trans.display("Monitor"); } mailbox to scoreboard.
    end
    endtask
end class
```

sampling of data

mailbox

* Note:-
Monitor and Scoreboard are option they are basically to cover every corner case ie get 99% code coverage. In our project duniption coverage 94% is enough and we have 98.3% hence we didn't bother.

Scoreboard :-

```
class scoreboard ;
    mailbox mon2scb;  // declaration.

    function new (mailbox mon2scb);
        this.mon2scb = mon2scb;
    endfunction                          } construction.

    task main;
        transaction trans; → object
        $spirat(:1)
        begin
        mon2scb.get(trans);   → you are getting the message from mailbox send by monitor to scoreboard.
```

verification of design by writting a code to check the logic of the design.

```
        if (((trans.a ^ trans.b) == trans.sum)
            && ((trans.a & trans.b) == trans.carry))
        else
            $error ("wrong Result");   → Refrence model.

        trans.display ("Scoreboard");
    end
    end task
endclass
```

* Note :- Refer to the note written under the monitor.

environment

```
`include "transaction.sv"
`include "generator.sv"
`include "driver.sv"
`include "monitor"
`include "scoreboard"
```
} why don't they both have .sv?

```
class environment;

  generator  gen;
  driver     driv;
  monitor    mon;
  scoreboard scb;
```
} all the four block in picture.

```
  mailbox m1; ⟶ gen2driv
  mailbox m2; ⟶ mon2scb.

  virtual intf vif;
  function new (virtual intf vif);
    this.vif = vif;
    m1 = new();
    m2 = new();
    gen = new(m1);          ⟶ gen2driv
    driv = new(vif, m1);
    mon = new(vif, m2);     ⟶ mon2scb.
    scb = new(m2);
  endfunction
```
} constructor are talking with interface

} both interact with interface.

~~book~~.

pleasure.

goes to the function.

```
task test();
  fork
    gen.main();
    driv.main();
    mon.main();
    scb.main();
  join
endtask

task run();
  test()
  $finish;
endtask.

endclass.
```

what is fork and join?
In SV, a fork/join splits the current thread into multiple parallel child threads, one for each statement in the block.

Note:-
in our code two is five changes.
1. we have include our component.
2. we have our event and due to which there is change in our argument for different class.
3. we have a addition task as pre-test when we test our design.
4. we have additional task post-test where we wait till the ended is triggred and we wait till the no. of expectation matches the no. of transaction.
5. in our test we run. pre_test() → test() → post_test()

test.

`include "environment.sv"

program test (intf i_inf);
↳ we are passing the actual interface

environment env; → declaring env.

initial.

begin ↳ we are creating an env within test.

env = new (i_inf);

env.run(); → running the mode run which triggers test() → which triggers every other model.

end

endprogram.

* Note:-

The only thing extra is that here we are giving the env.gen.repeat_count = 200; rather than hardcoding it in other places.

top module:

`include "interface.sv"

`include "test"

module tbench_top;

intf i_intf();

test t1 (i_intf); } initiating the both module.
↳ passing actual interface.

half_adder h1 (

.a (i_intf.a),

.b (i_intf.b),

.s (i_intf.sum),

.c (i_intf.carry)

); } DUT.

initial begin:
$dumpfile ("dump.vcd"); $dumpvars; } for getting waveform
end.

endmodule.

* Note :-
1. in our top we have just added clock signal and basic on and off reset.

⇒ We have additional coverpoints.

class fib_cov;
fib_trans trans = new();
covergroup cov_inst;
option.per_instance = 1;
ST: coverpoint trans.start { bin st = {0,1}; }
↳ FSM is allowed to be 0 or 1
I : coverpoint trans.i { bins i = { [0:31] }; } → max value 31 as 5 bits
RY: coverpoint trans.ready { bins st = {0,1}; }
↳ FSM is allowed to be 0 or 1
DT: coverpoint trans.done_tick { bin dt = {0,1}; }
↳ FSM is allowed to be 0 or 1
F: coverpoint trans.f { bins f = { [0: 1346269] }; }
max value of 14 bit can be 1346269.
for i = 31 f = 1346269. which in binary is
0001 0100 1000 1010 1101 1101
endgroup.

cover group.

```
function new();
  cov_inst = new;
endfunction.
```
} created the.

```
task main;
  cov_inst.sample();
endtask

endclass.
```