SYSTEM VERILOG ESSENTIALS NOTES (UDEMY)

System Verilog

Generator class → Generate stimulus for DUT and send driver class.

Ports

- Global signal (clock & reset) ✓
- control signal (readdr, waddr, WE, OE)
- Data signal (rdata, Wdata)

testbench.sv (Initial block)

```
module tb();
reg a = 0;
initial a = 1;

initial
begin
a = 1;
#10
a = 2;
end

endmodule
```

initial block start of simulation is executed.

DUT (Design under test)

Usage of Initial block

```
module tb();

  initial begin
    $dumpfile("dump.ved");   } → Analyze waveform
    $dumpvars;
    end
  endmodule.
```

```
timescale 1ns/1ps.
module tb();
  reg clk;
  reg rst;  → reg [3:0] temp;
    initial begin
      clk=0;   or 1'b0;
      rst=0;   } → Global signal.
temp=4'b0000; ← end
#10;
temp= 4'b0011;
```

2. random signal.

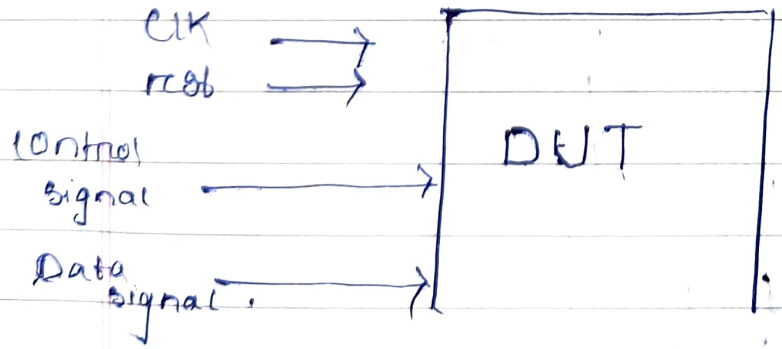
3. System task at the start of simulation.

```
initial begin
$monitor ("Temp: %0d at time : %0t", temp, $time);
end
```

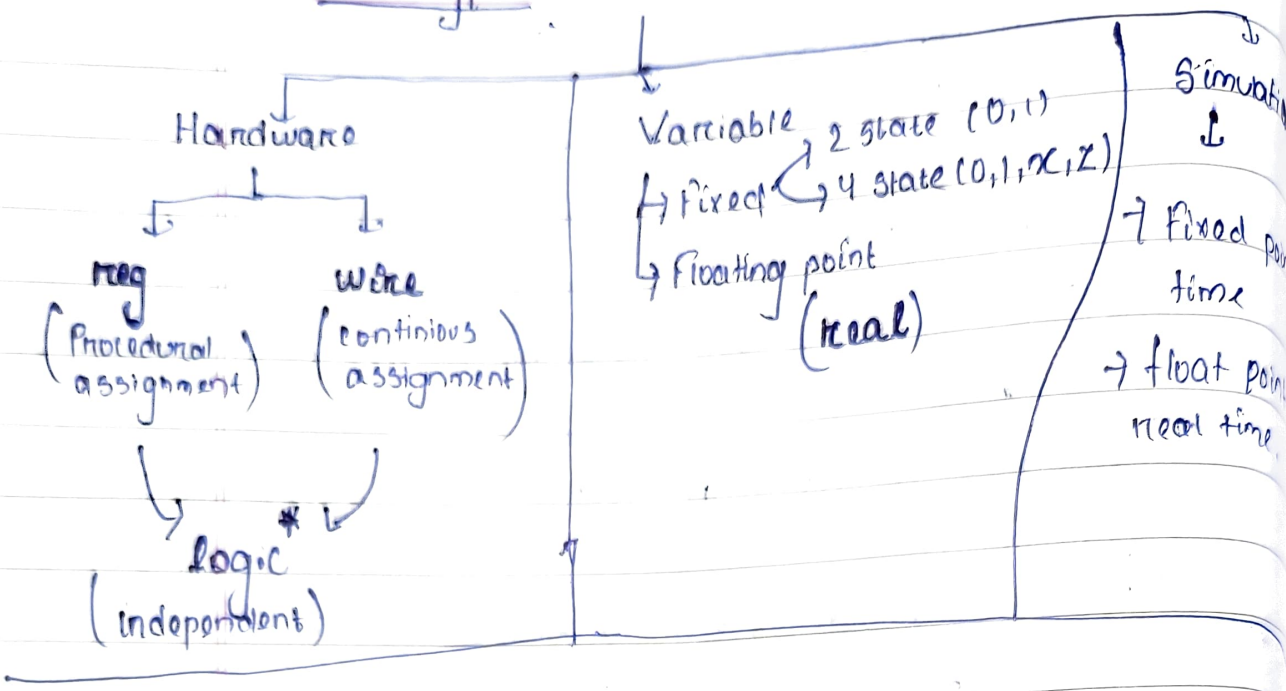
4) Analyzing value of variable on console.

5) Stop simulation by forcefully calling

```
initial begin
  #200;
  $finish();
end
endmodule.
```



Datatypes



Variable

floating point (real)

fixed point

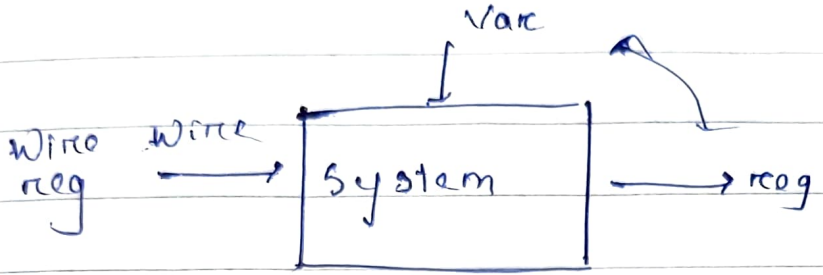
2 state (0,1)

4 - state (0, 1, x, z)

signed

unsigned

signed	unsigned
8-bit → byte	bit [7:0]
16-bit → shortint	bit [15:0]
32-bit → int	bit [31:0]
64-bit → longint	




```
integer count;
always @ (posedge clk)
    begin
        count <= count + 1;
assign y = count;
```

Declaring Array

```
module tb;
bit arr1 [8];
bit arr2 [] = {1,0,1,1};
initial begin
  $display ("size of arr1 : %0d", $size(arr1));
  $display ("size of arr2 : %0d", $size (arr2));

 end
endmodule
```

```
module tb;
bit arr1 [8];
bit arr2[] = {1,0,1,1};

initial begin

    $display ("size of arr1 : %0d", $size(arr1));

    $ display ("value of first element of arr1 %0d", arr2[0]);
    arr1 [1] = 1;
$display ("value of first element of arr2[0] : %0d", arr1[1]);

$display ("value of all element of arr2 : %0p", arr2);

end
endmodule.
```

%0p → Prints all value
↳ '{1'b1, 1'b0, 1'b1, 1'b1}'

Array Initialisation

- Unique value: arr[] = {1, 2, 5, 4, 5}
- Repetitive value: arr[] = {6{1}}
- Default value: arr[7:0] = {default:0};

Uninitialised array

bit arr[5];

- {0,0,0,0,0}

(zero)

logic arr[5];

{x, x, x, x, x}

(don't care)

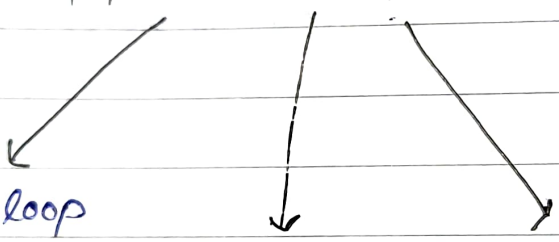
int arr[5] = '{5{0}};

o/p - {0,0,0,0,0}

int arr[10] = '{default : 2};

o/p - {2,2,2,2,2,2,2,2,2,2}

Repetitive Operations



for loop

repeat

for each

for loop

```
module tb;
 int arr[10];
 int i;
 initial begin
   for(i=0; i<9; i=i+1) begin
     arr[i] = i*i;
   end
   $display("arr: %0p", arr);   // {0,1,4,9,16,..81}
   end
endmodule
```

Uninitialised array

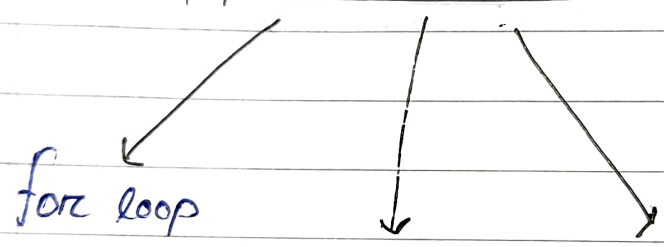
bit arr[5];
- {0,0,0,0,0}
(zero)

logic arr[5];
{x, x, x, x, x}
(don't care)

int arr[5] = '{5{0}};
O/P - {0,0,0,0,0}

int arr[10] = '{default : 2};
O/P - {2,2,2,2,2,2,2,2,2,2}

Repetitive Operations



for loop

```
module tb;
 int arr[10];
 int i;
 initial begin
   for(i=0; i<9; i=i+1) begin
     arr[i] = i*i;
   end
  $display("arr: %0p", arr);   // {0,1,4,9,16,..81}
  end
endmodule
```

```
module tb;
int arr[10];
int i;
foreach [arr[j]] begin
        arr[j] = j;
        $display ("%0d", arr);
```

Foreach

```
module tb;
  int arr[10]
  initial begin
    foreach (arr[j]) begin   // no need to write i++ or i=0
      arr[j] = j*j;          // don't have to initialize j
    end
  $display ("%0p", arr);
  endmodule.
```

repeat

```
module tb;
  int arr[10];
  int i;
  initial begin.
  repeat (10) begin
    arr[i] = i*i;
      i++;
  end
    $display ("%0p", arr);
    end
    endmodule.
```

Array operation P1 : copy

```
module sv1;
    int arr1[5];
    int arr2[5];

initial begin
    for(int i=0; i<5; i++) begin
        arr1[i] = i*5;
    end
    arr2 = arr1;
    $display("%0p", arr2);
    end
endmodule.
```

// Compare

```
module sv1;
int arr1[5] = '{1,3,6,7,1};
int arr2[5] = '{1,3,6,7,1};
int status;
    initial begin
    status = (arr1 == arr2);
$display("status : %0d", status);
end
endmodule.
```

status = 1

Dynamic Array

```
module sv1;
    int arr[];                    initial begin
for (int i; i<5 ; i++) begin
        arr[i] = 5*i;
    end
    arr = new[30] (arr);
    $display("arr: %0p", arr);
and
endmodule.
```

(adds first 5) 0 5 ... to new 30

```
module tb;
  int arr[];
  int arr[30];
  initial begin
    arr = new[5];
    for (int i=0 ; i<5 ; i++) begin
        arr[i] = 5*i;
    end.
    arr = new[30](arr)
    $display("arr : %0p", arr);
     farr = arr;
   $display("farr : %0p", farr);
```

0, 5, 10, 15, 20, 25, ...

```
module tb;
  int arr[$];        // queue
    initial begin
    arr = {1,2,3}
    $display("arr : %0p", arr);

    arr.push_front(7);
    $display("arr : %0p", arr);   // 7 1 2 3   o/p
    arr.push_back(9);
    $display("arr: %0p", arr);    // 7 1 2 3 9   o/p

                  index   element to be inserted
    arr.insert(2, 10);
    $display("arr : %0p", arr);   // 7 1 10 2 3 9

    int j = arr.pop_front();      // 7 is popped.
    $display("arr : %0p", arr);   // 1, 10, 2, 3, 9
    $display("value of j : %0d", j);  // j=7

    j = arr.pop_back();
    $display("arr : %0p", arr);   // 1, 10, 2, 3
    $display("value of j: %0d", j);  // 9

    arr.delete(1);                // 10 is at 1
    $display("arr : %0p", arr);   // 1 2 3
    end

endmodule.
```

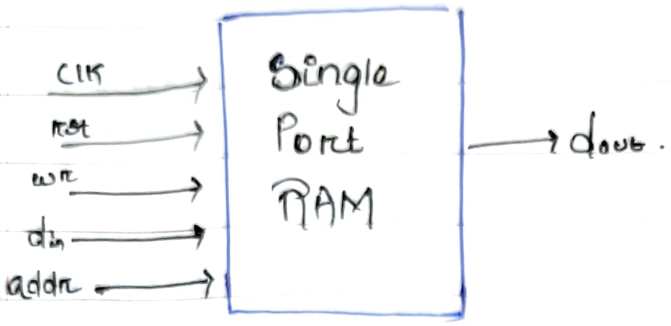
Usage of fixed size array.

```
class transaction;
rand bit [7:0] din;
randc bit [7:0] addr;
bit wr;
bit [7:0] dout;

    constraint addr_c { addr > 10; addr < 18;};

endclass

class generator;

transaction t;
integer i;

    task run();
      for (i=0; i<100; i++) begin
        t = new();
        t.randomize();
    end
    endtask
    endclass
```

Next pag →

```
class scoreboard;
  bit [7:0] arr[256] = '{default:0};
  transaction t;
  task run();
.....
    if (t.wr==1'b1) begin
      arr[t.addr] = t.din;
    $display ("[sco] : Data stored din : %0d addr : %0d",
                                    t.din, t.addr);
    end
.
if (t.wr == 1'b0) begin
      if (t.dout == 0)
        $display ("[sco] : no data written at this location
                                    test passed");
      else if (t.dout == arr[t.addr])
        $display ("[sco] : valid data found Test case
                                    Passed");
      else
        $display ("[sco] : Test failed");
      end
  end
  endtask
endclass.
```

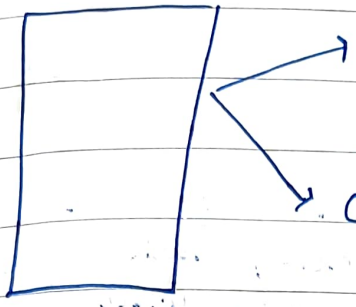
Verification Plan



RAM Verification plan

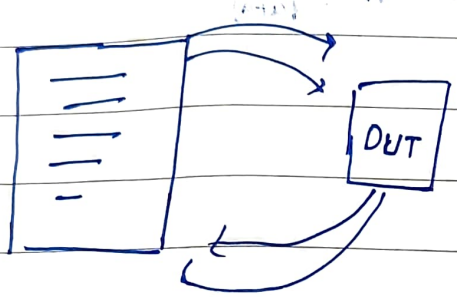
	Test case	Description	Feature covered
1	rst_high	Default values of dout when reset is asserted	This behaviour of dout at the start of operation when rst is high for few
2	rst	reset is asserted during read and write transaction	Dout must hold its initial value whenever reset is asserted in the middle of ongoing Transaction.
3	Wr	Wr is asserted after reset is deserted	Verify valid data store in the memory during write Transaction.
4	Wr	Wr is asserted when reset is deasserted	dout must return valid value store during prev. write transaction.
5	Wr_low	Wr is deasserted when reset is deasserted	dout must return value store during previous write transaction.

wb_low	wb is asserted when reset is asserted	dout must stay at zero.
wb_low	wb is deasserted when rst is deasserted	reading data from address without valid data, must return zero.

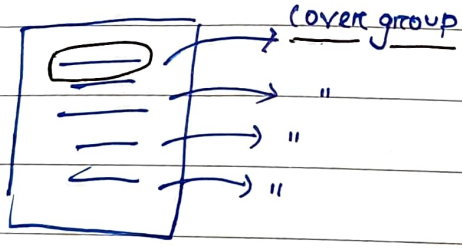


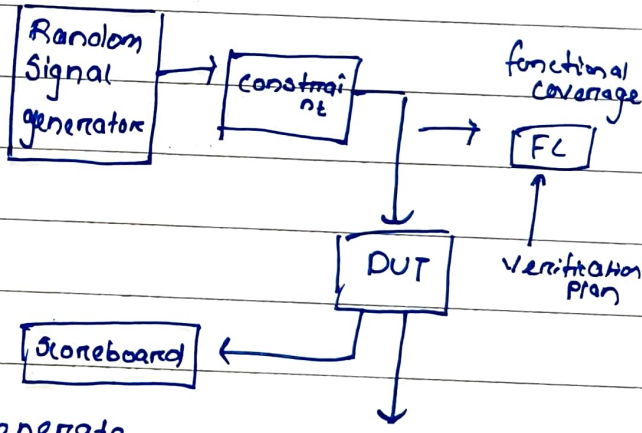
Directed Test

Constraint random test.



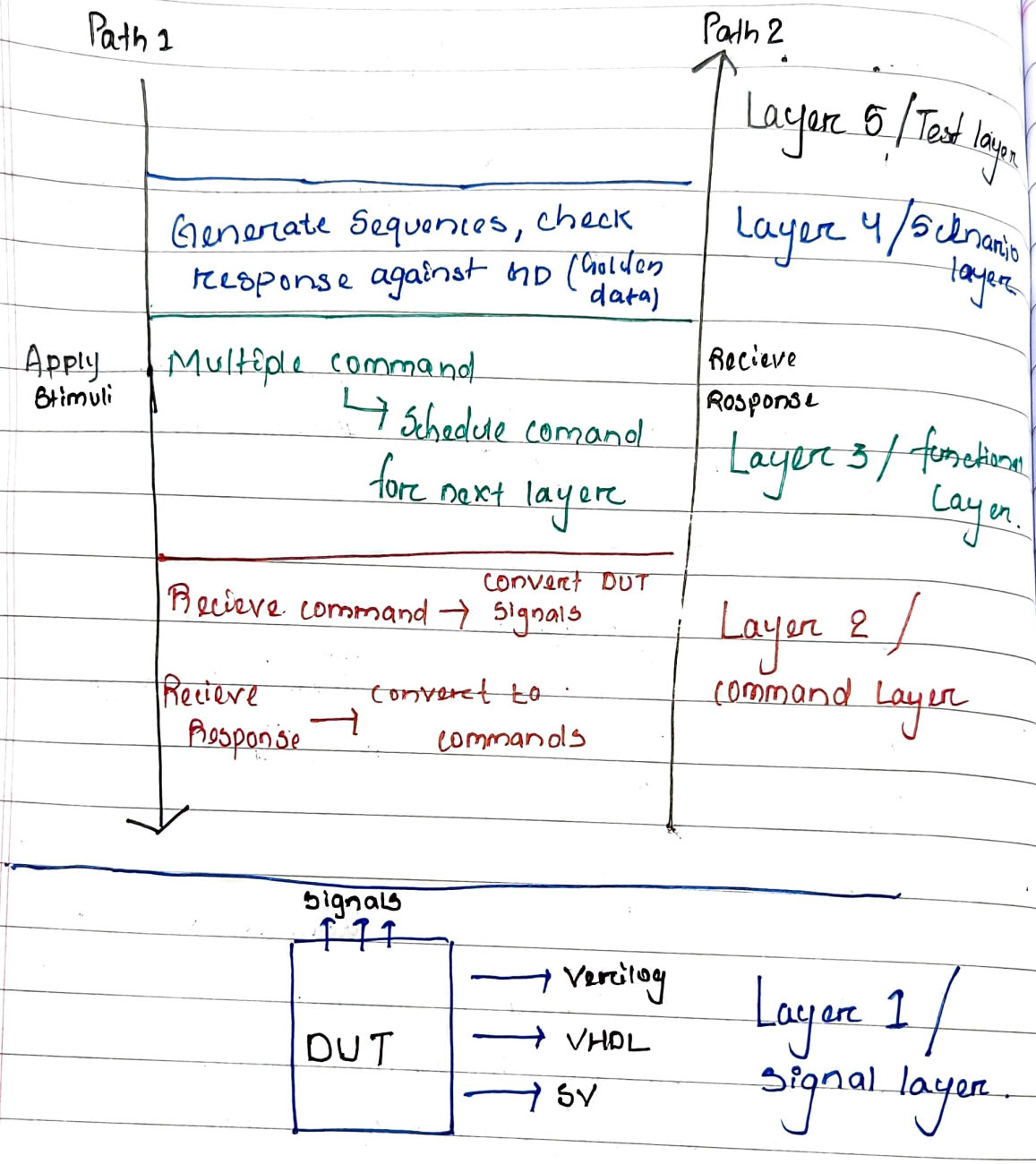
Individually take test case & analyse response.





✓ It is a technique used to generate randomize test case with specific constraints to ensure that the generated input stimuli meet certain design requirements.

Layered Testbench Architecture.



ayer 5

Upper layer will control simulation until all the sequences are applied to DUT and we also recieve response from DUT.

Layer 4

Generate sequences to verify specific feature of DUT
+
checking response received against GD

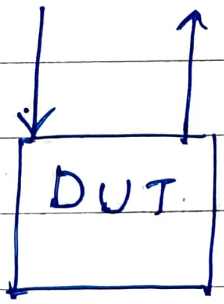
Individual command → Drive signal

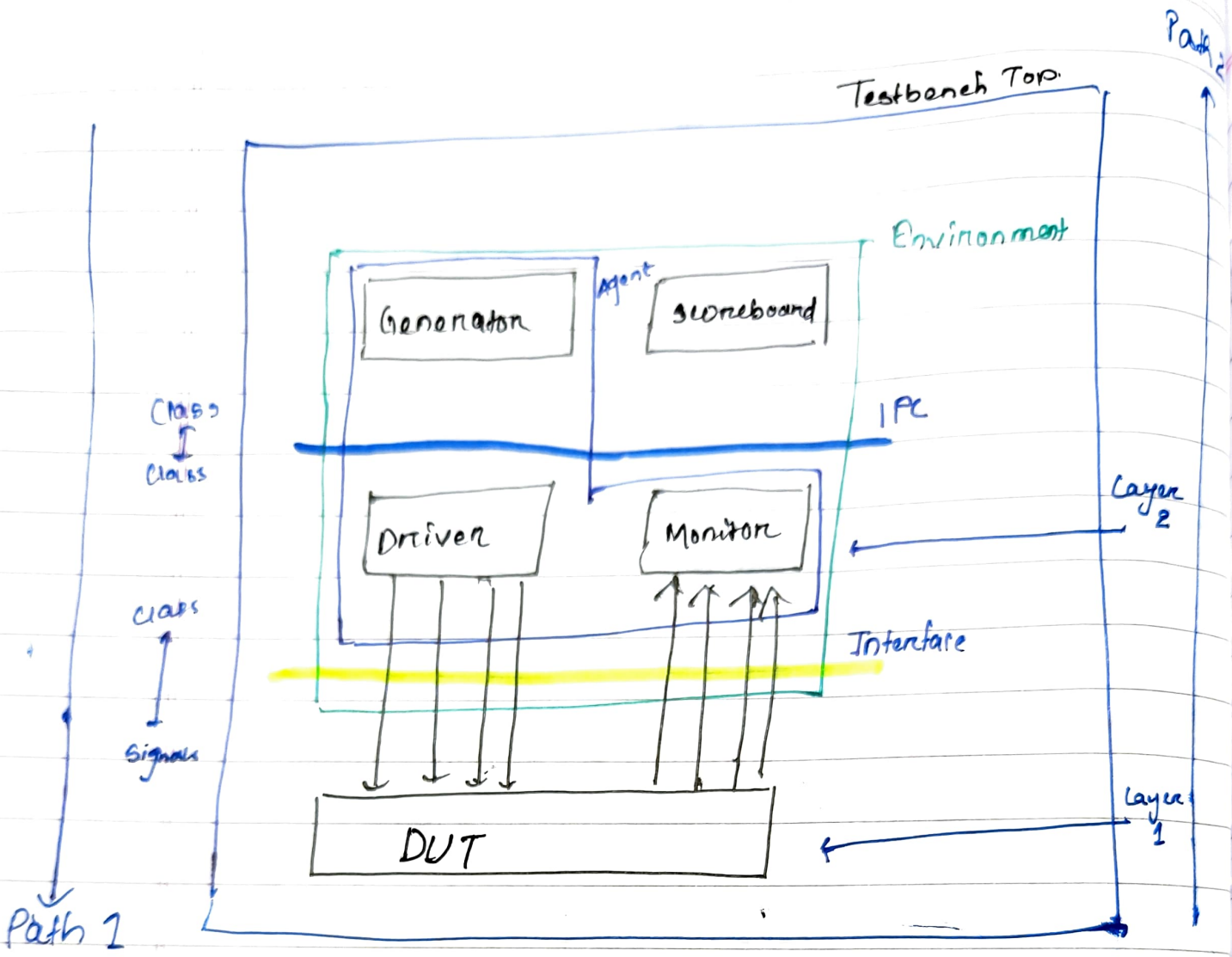
Layer 3

Scheduling individual command for specific sequence.

Layer 2

Individual command → Drive signal
Receive response → command





Transaction:- Contain Variable for IP/OP Port present in DUT

Generator: Generate random stimulus and send it to driver using IPC

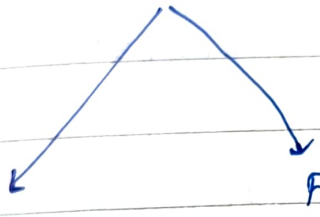
Driver:- Recieves stimulus from generator & Trigger respective signals of DUT with the help of Interface.

Monitor: Receive response from DUT and send it to Scoreboard using IPC.

Scoreboard: Compare response of DUT with expected/ golden data.

SECTION 5

Class Method



Task

→ Support timing control
@(posedge clk)
#10

→ Multiple O/P port

Function

→ Do not support timing control

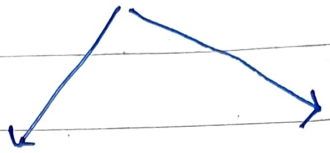
→ Do not support O/P port.

Using function

GITHUB Function & Task.

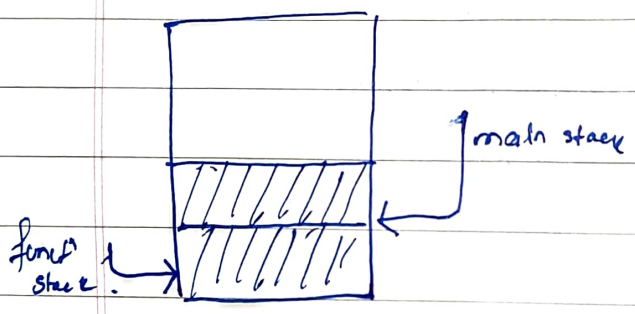
~~module tb;~~

Task & function



Pass by value Pass by reference.

```
task add (int x, int y);
    x = x+5;
    sum = x+y;          add (a, b);
endtask
```

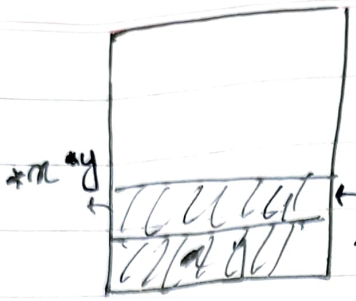


Pass by value

Even though we are changing value like x=x+5, it is local to function. This will not be reflected in main.

Pass by reference.

```
int x;
int y;
task add (ref int x, y)
    x = x + 5;
    y = y + 5;
endtask
```



As we change the value of x, y, the change will be reflected in Testbench top.

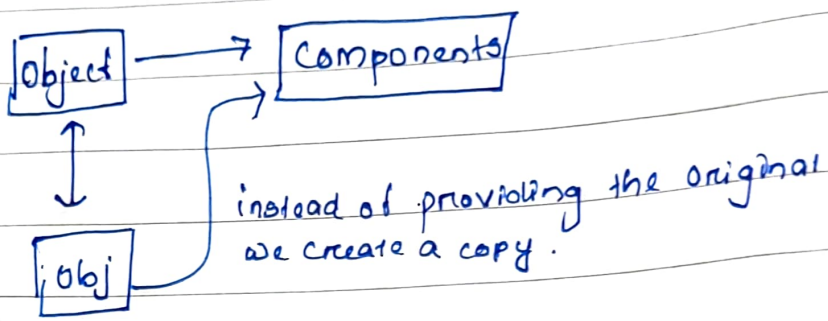
ref int a[] (You want to change later)
a[] → updated - after Processing

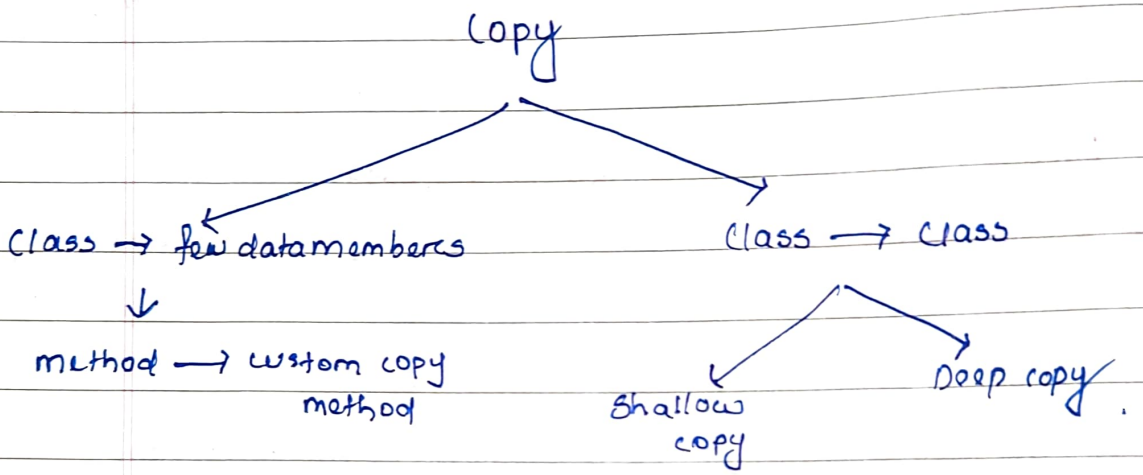
Scalars → Pass by values (creates a local copy of x,y)

Arrays → Pass by reference.
- a[] → initial value.
- const ref int a[] (can not change)

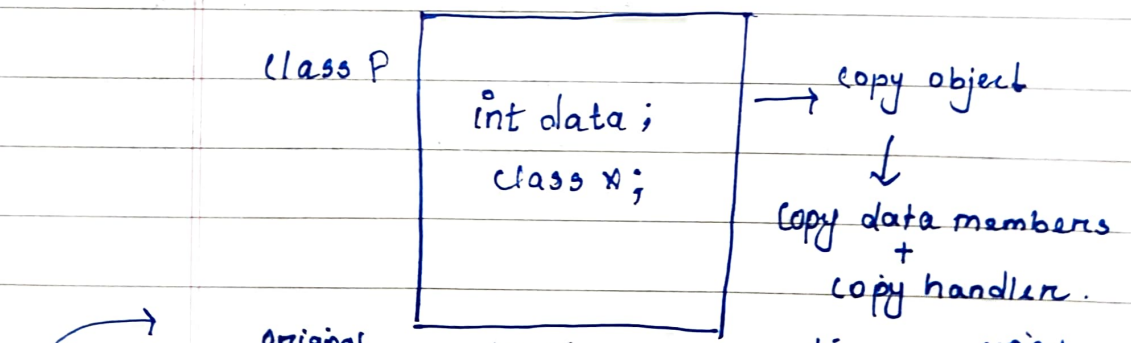
Function	Task
→ Do not support timing control.	→ Support timing control.
→ Multiple I/P single return value.	→ Multiple input/output.
	→ Use ref, automatic for array
Use: print value of data members, initializing values of variables, Time independant expression, return data from class.	Use: Time dependant expressions, Shedding processes in class.

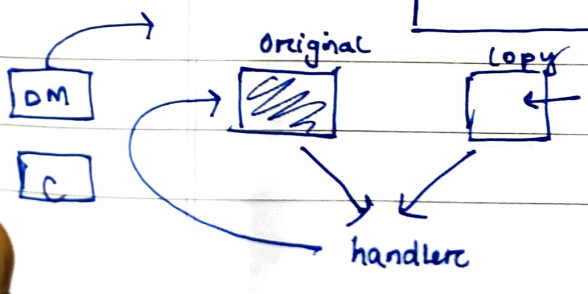
Strategy to copy data.

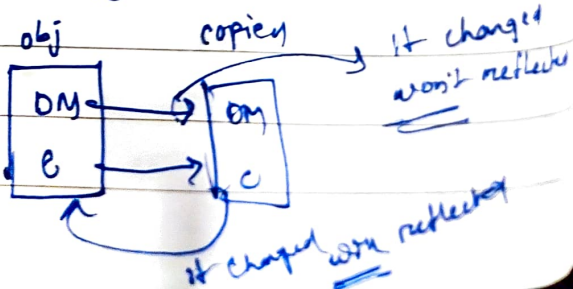




Shallow Copy.

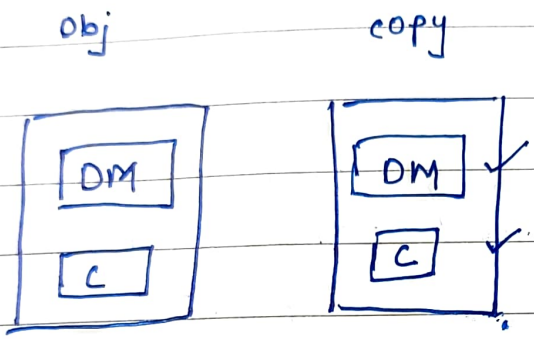






A shallow copy in SV is a copy of an object where only the references to the data members are duplicated, not the actual data itself. This means changes to data members in the copied object will affect the original object as well.

Deep Copy



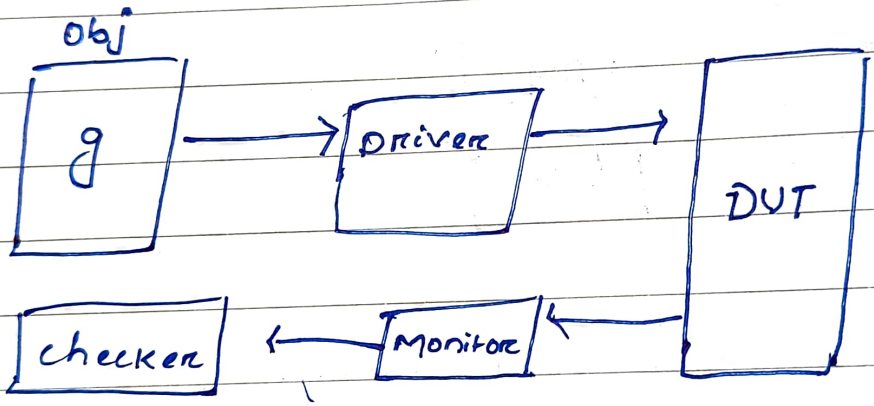
Here independently we are creating DM & C for copy obje

Randomization

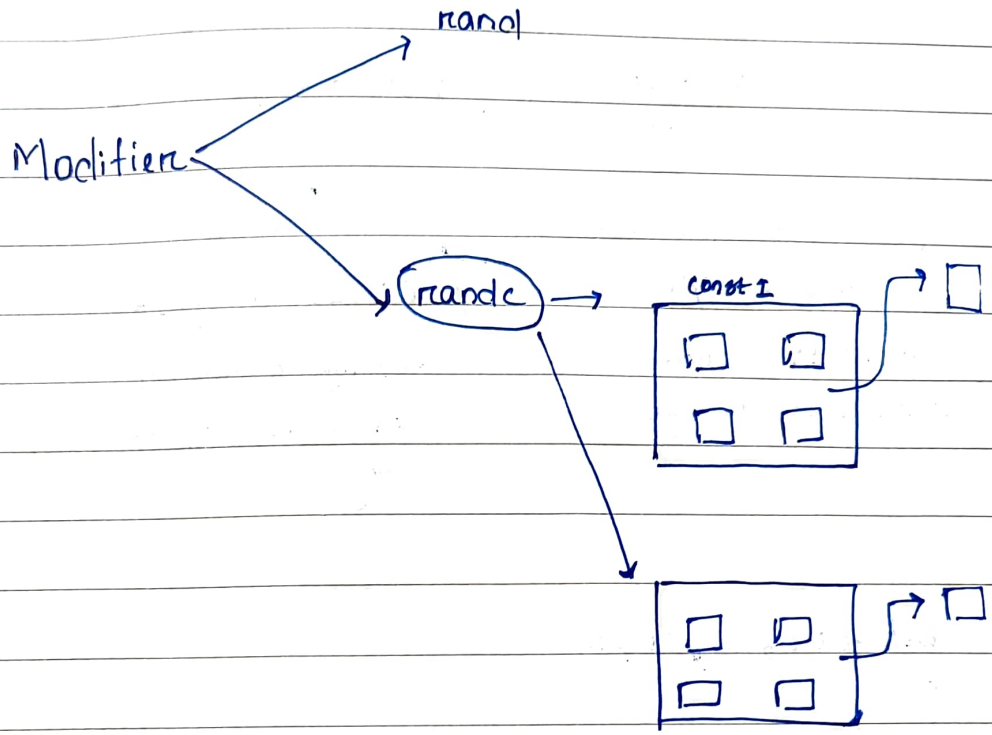
Generator class → Generate a stimuli to dut and send to Driver class.

rand vs randc

(↓ won't get repitation of any value.)



10



If we change the space the previous value won't affect for the second. (see code 10 from Randomization).

Weighted Distribution.

```
class temp;
rand bit wr;  → wr=1 , wr=0
  constraint wr_const {
        wr dist {0:=2  1:=3};
    }
endclass
```

```
wr dist {0:= 10; 1:=90};
```

100

$P(0) = \frac{10}{100} = 0.1$

$P(1) = \frac{90}{100} = 0.9$

Wn dist {0 :/ 10; 1 :/ 90; };

$P(0) = \frac{10}{100} = 0.1$, $P(1) = 0.9$

:=	:/
Equal weight to all the values within the range	Divides weight equally between values within the range

2-bit → 00, 01, 10, 11

↑

Sll dist {00 := ; [1:3] := 60}.

{0 := 10; 1 := 60; 2 := 60; 3 := 60; }

$P(0) = \frac{20}{190}$

$P(1) = \frac{60}{190}$

$P(2) = \frac{60}{190}$

$P(3) = \frac{60}{190}$.

Sol dist { 0 :/ 10; [1:3] :/ 60};

{0 :/ 10 ; 1 :/ 20; 2 :/ 20 ; 3 :/ 20 } ;

$P(0) = \frac{10}{70}$

$P(1) = P(2) = P(3) = \frac{20}{70}$.

:/ (will divide range equally).

Operators

Implication ($\rightarrow$)

$(x = 1) \rightarrow (y = 1)$

$x \rightarrow 1 \quad y = 1$

$x \rightarrow 0 \quad y = 0, y = 1$

equivalence ($\leftrightarrow$)

$x = 0, y = 0$

$x = 1, y = 1$

If else (if)

if ( ) { }

else { }

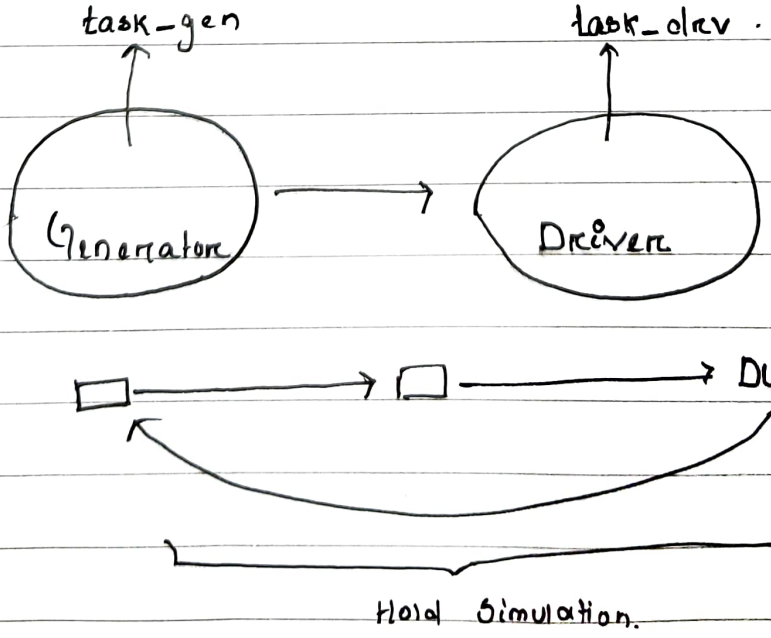
curly bracket used in SV.

behaves as an implication).

IPC (Interprocess Communication)

- **Event** (convey message between classes)
  - → trigger
  - @ edge sensitive blocking @()
  - wait level sensitive non blocking wait()
- **Semaphores** (access resources of tb_top)
  - get
  - put
- **Mailbox** (Send transaction data b/w classes)
  - get
  - put

Executing Multiple Process.



fork Join

Allow multiple process to execute in Parallal

initial begin

fork

} → Processes

join

Understanding Semaphore.

It is used to manage concurrent access to shared resources. Semaphore allow tasks to signal each other & control access to critical section