Date:- 22/5/2025

System verilog [SV]

Introduction:-

- It's an HDVL language, HDVL refers to the Hardware description cum verification language.
- Helpful for learning and applying in verification methodologies.
- Base to learn 'UVM' universal verification methodology.

What is System verilog?

⇒ extension of verilog.

⇒ Bulk of verification is based on "OpenVera" language

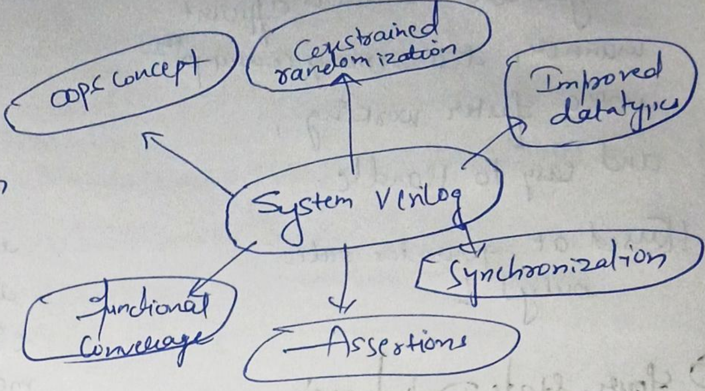
⇒ standerdised IEEE 1800-2005 enhancement of IEEE 1364 verilog-2001

⇒ Has features from verilog, VHDL, C, C++, nowadays python also

⇒ gives verification environment can be written using system verilog concept allows reusability

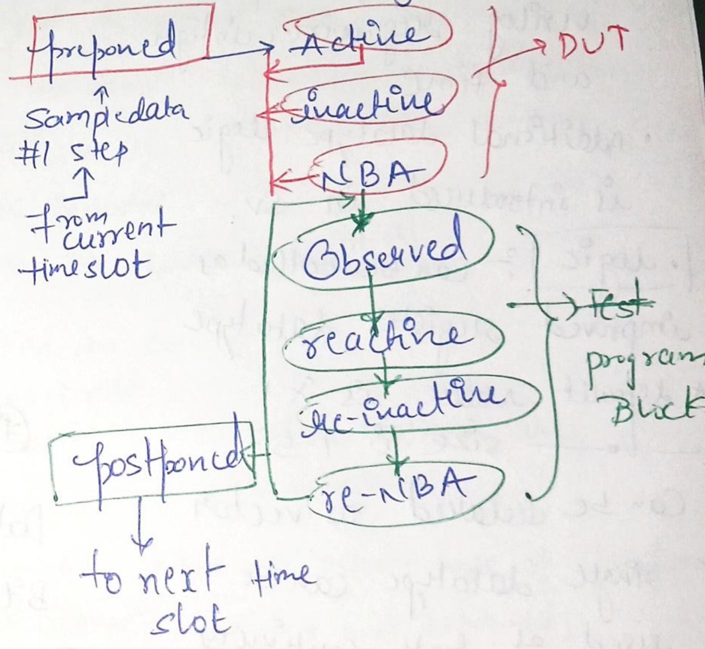
⇒ testbench codes in 'SV' used to check functionality correctness of DUT by generating stimulus & driving predictable input and captured output, compare output

Features of System verilog:-



Regions in System Verilog:-

introduces '4' more regions



Datatypes:-

• 4 state type [0,1,x,z]	• structures
• 2 state type [0,1]	• unions
• real	• strings
• Arrays	• enumerated
• user defined	• class

* why 2-state in sv?

in generation stimulus efficient manner, less memory consump^th more faster working, and easy to Handle.

![used at generator side only]!

four state Datatypes:-

- allowed values are 0,1,x,z
- followed datatypes from verilog reg, wire, integer and time
- Additional datatype "logic" is introduced in SV.

logic :- can be called as improved register datatype

* default value is 'x'
  size is 1-bit
* can be declared as vector
* single datatype can be used as both continuous and procedural statements

limitation of logic

- doesn't support multi driver conditions

ex:-
```
reg [7:0] a;
initial begin
   a=1;   → 32 bit truncated value to 8 bit
   a='1;  → All pos^n are 1 value
```

user can define size for logic using vector form

ex:- logic vector
```
module and-gate (input logic a,b,
* output logic c);
assign c=a&b;
endmodule
   (or)
always @ (*)
   c=a|b;
endmodule
```
but
```
c=a&b; }
c=a|b; } x not allowed in logic
```

2 state Datatype:-

Datatype	size	type signed/unsigned	ref value All have zero
Bit	1-bit	unsigned	0
Byte	8	signed	—"—
int	32 bit	—"—	—"—
short int	16	—"—	—"—
real	64	—"—	—"—
shortreal	32	—"—	—"—
realtime	64	—"—	—"—

* reg, wire, bit & logic can
be declared as vector.
[their size is 1 bit]

integer & int
vs
* 4 state   2 state
'x'   def value   '0'

* bit [7:0] a; byte b;
unsigned   signed.
0 to 277   -128 to 127

* real & realtime
no difference, interchangable
used in sim purpose

* bit a;
a = 1'bx;
gives zero but not
'x'

```
module ex1;
 int a;
 int unsigned b
  bit signed [7:0] c;
 initial begin        → (-1)
  a = -32'd127;
  b = '1 ; c = 'b;
 end
endmodule
```

```
* int a;
 logic [31:0] b = 'z'
 initial
  begin
 a = b;
 b = 32'b123a-5678;

 if($unknown(b))
 $display ("b is unknown");
 else
 $display ("b is known.");
 end
endmodule
```

(#) Real & void type:-

used in functions, to suppress return type.,

• real included from verilog, real same as double in 'c'.

Addition to SV :-
⇒ shortreal
⇒ realtime ; ! real & realtime are interchangeable

void
↳ can be used as return type of functions to indicate nothing is returned.

Q. diff b/w fn in verilog vs fn in system verilog

ex:- function void display;
    $display ("Hello")
    ~~return value;~~ ← this
    //gives error
    endfunction

ex:- void can also be used in typecasting & to remove return type

//driving x&z to byte

byte [7:0] byte-with-x;
—"— [7:0] ——"—-z;
initial begin
byte-with-x = 8'b1x0x_1x0x;
——"——-z = 8'b1z0z_1z0z;
    gives error
→ //Declare as logic to print

(.sv) file extension

Q. write a sv code:-
ⓐ declare all the following datatypes logic bit, byte, int, shortint, longint..
ⓑ print the default value, size. of each datatype
ⓒ drive the 'x' & 'z' in byte, and print this value

Ans:- module dfdt_pgm;
bit a;
logic b;
byte c;
shortint d;
int e;
longint f;
initial begin
$ display (bit %0d, %0b, $bits(a),a);
——"—— (logic df ~~[illegible]~~ size = %0d, value = %b, $bits(b), b);

//ly for others
end
endmodule

ex:- function void display;
    $display ("Hello")
    // return_value; ← this
    // gives error
    endfunction

ex:- void can also be used in typecasting & to remove return type

(.sv) file extension

Q. write a sv code:-
ⓐ declare all the following datatypes logic, bit, byte, int, shortint, longint ..
ⓑ print the default value, size of each datatype
ⓒ drive the 'x' & 'z' in byte, and print this value

Ans:- module dfdt_pgm;
    bit a;
    logic b;
    byte c;
    shortint d;
    int e;
    longint f;
    initial begin
    $display(bit %0d, %0b, $bits (a), a);
    —"— (logic ~~dy~~ size %0d, value = %0b, $bits(b), b);

    ||| for others
    end
    endmodule

//driving x & z to byte

    byte [7:0] byte_with_x;
    —"— [7:0] —"—z;
    initial begin
    byte_with_x = 8'b1x0x_1x0x;
    —"—z = 8'b1z0z_1z0z;
    gives error

→ //Declare as logic to print

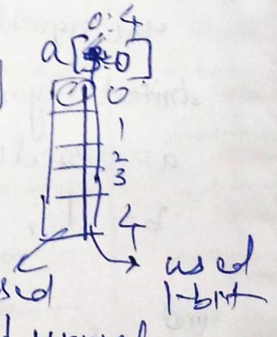
Concept:- Arrays

can be vector
- packed
  - size fixed
  - can't be dynamic
- unpacked
  1) fixed
  2) dynamic
  3) queues
  4) Associative

enum array user defined

differences

	packed	unpacked
Decl^n	dt [size] name;	dt name[size];
support	bit, wire, logic & reg	All datatypes
memory Arch	reg [3:0] a | x | x | x | x | ↓ each cell one bit size	reg a[4:0]
	more efficient no wastage of memory	less efficient wastage can be seen

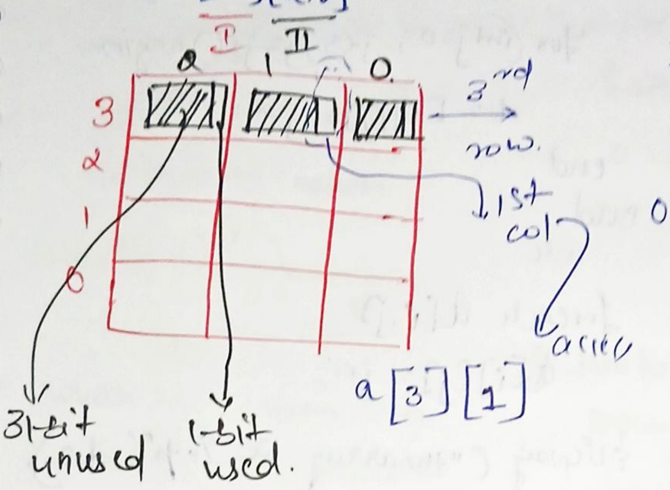


31-bit unused.

packed	unpacked.
not flexible	flexible

visualizing the memory

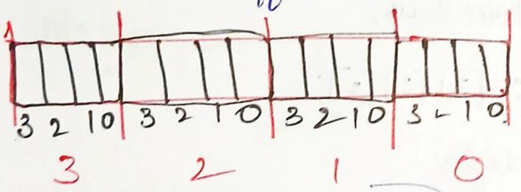
bit a[3:0][2:0]



Same in case of packed like

bit [3:0] [3:0] b;

will be different



acces,

b[1]; entire first cell of 4 bits

using foreach loop

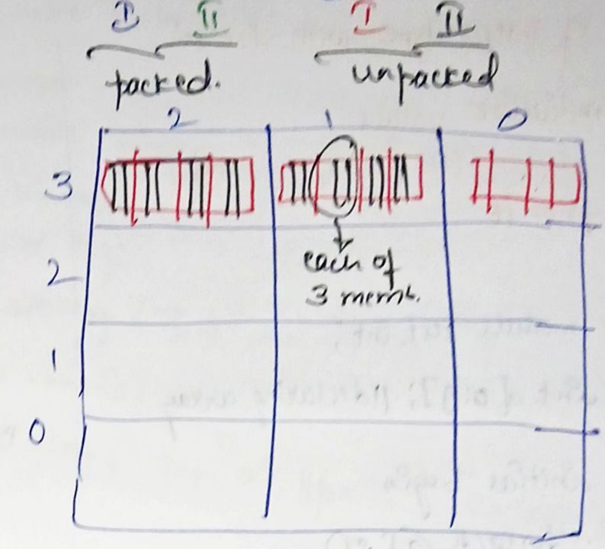
```
for each a[i]
a[i]= 5*i;
etc.,
```

using for loop

```
for(i=0; i<4; i=i+1)
a[i]= {$random} % 20;
```

(11) Concept mixed memory :

bit [8:0] [2:0] a[3:0] [2:0];



I → packed 1st dimension.

II → unpacked 2nd dimension

I → unpacked 1st dimension

II → unpacked 2nd dimension

(11) Concept: initialization of the Array

```
int a[3:0];
initial begin
  a[3] = 5;
  a[2] = 10;
  a[1] = 15;
  a[0] = 20;
end
```

listing Method

or

o/p, a → '{5,10,15,20}

or

a = '{5,10,15,20}

or

or

Q. program in (SV) to
a) declare fixed unpacked array of integer type with size '10'
b) initialize values to {1, 2, ..., 10}
c) print it

Ans:-
```
module test_int;
  int a[0:9]; //declaring array
  initial begin
    foreach (a[i])
      a[i] = i+1;
    $display ("the array a is:%p", a);
  end
endmodule
```

to print by element-wise
```
foreach (a[i])
$display ("the element a[%d]=%d",
      ...i, a[i]);
```
in foreach loop.

Q. write a code in (SV) to
a) print & declare a 2-D unpacked fixed array of integer rows, columns = 5,5;
b) initialize them
c) print them

Ans:- Code:-
```
module dd_unpacked_array;
  int d2[5][5];
  initial begin
  for (int i=0; i<5; i=i+1) begin
    for (int j=0; j<5; j=j+1) begin
       d2[i][j] = j+1;
  end
end
   or
  foreach d2[i,j]
     d2[i][j] = j+1;
  $display ("the array is %p", d2);
end
endmodule
```

Q. same with 3-D unpacked fixed integer type array

```
module three_d_up;
  int c3[3][3][2];
  initial begin
   foreach (a[i,j,k]) begin
   a[i][j][k] = k+1;
   $display ("the a[%d][%d][%d] = %d",
       i, j, k, a[i][j][k]);
  end
emodule.
```