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

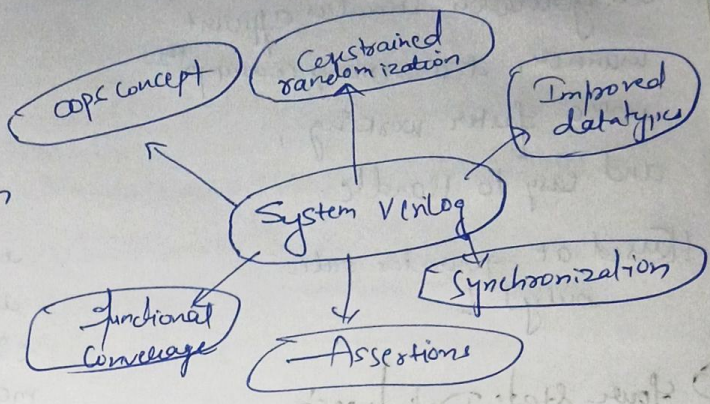
⇒ standerdised IEEE 1800-2005 enhancement of IEEE 1364 verilog-2001

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

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

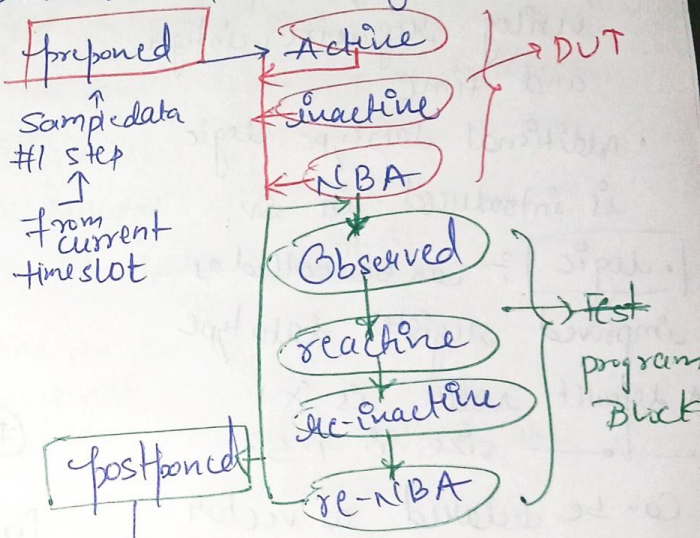
⇒ testbench codes in 'SV' used to check functionality correctness of DUT by generating stimulus & driving predictable input and captured output, compare output

Features of System Verilog:-



Regions in System Verilog:-

introduces 4 more regions



Datatypes:- [0, 1, x, z]

- 4 state type
- 2 state type [0, 1]
- real
- Arrays
- user defined
- structure
- union
- strings
- enumerated
- class

* why 2-state in sv?

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

[used at generator side only]!

(#) 4 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
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; } * not allowed in logic
c = a|b;

(#) 2 state Datatype:-

Datatype	size	signed/unsigned type	default value all have zero
Bit	1-bit	unsigned	0
Byte	8	signed	0
int	32 bit	〃	〃
short int	16	〃	〃
real	64	〃	〃
shortreal	32	〃	〃
realtime	64	〃	〃

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

integer & int vls

* 4 state 2-state
  'x' no value '0'

* bit [7:0] a; byte b;
  unsigned signed.
  0 to 255 -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'b03x-5678;
  if($unknown(b))
  $display ("b is unknown");
  else
  $display ("b is known.");
  end
  endmodule.

(11) 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 interchangable

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

If 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

(.sv) file extension

Q. write a sv code:-
(a) declare all the following datatypes logic bit, byte, int, shortint, longint..
(b) print the default value, size of each datatype
(c) 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(5), b);
// for other
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