SASKEN

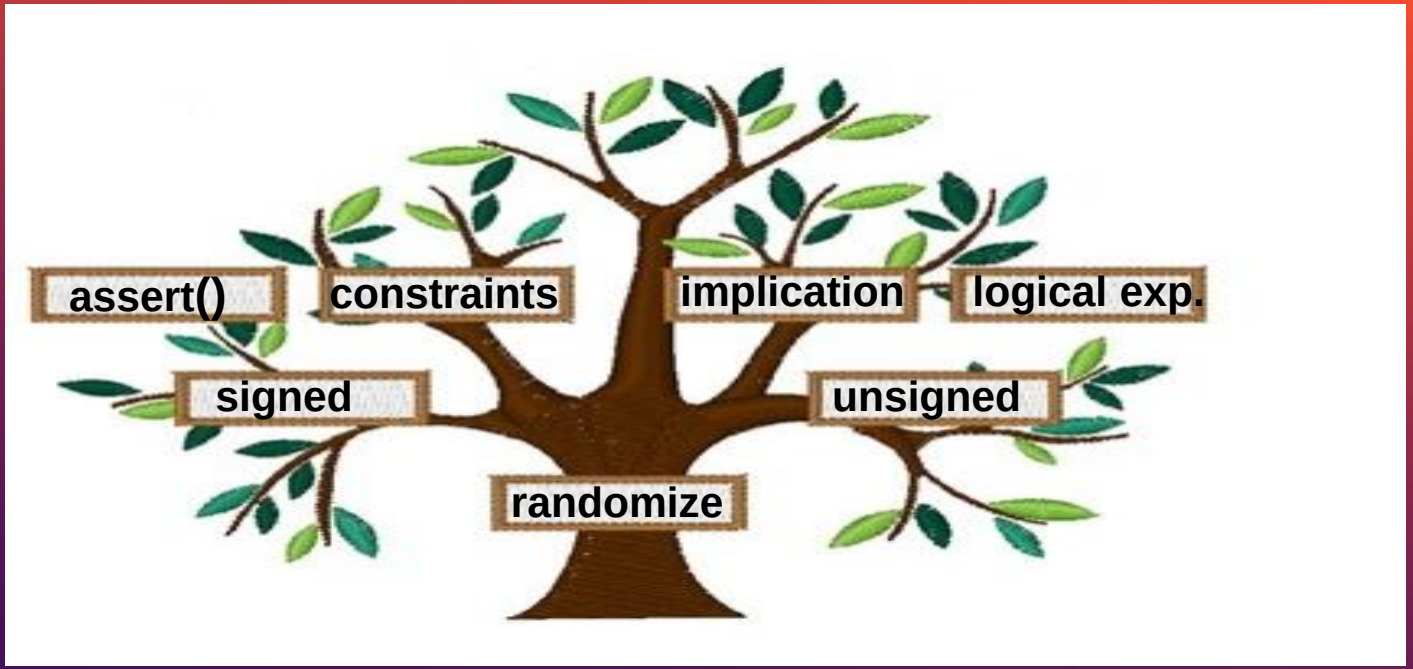
Things to know about Randomization in VIPs

Things to know about Verilog System Verilog Randomization in Verification IPs (VIP)

SASKEN

Introduction

- Randomization is a kind of “automation”
- Stimulus generation; Directed, Random and Constrained-Random
- “constraints” provide more controllability and hence specifications are so closer
- Much care should be given when randomizing signed numbers



Less Directed Stimulus

SASKEN

plenty of opportunities . . . 8 9 10

```
constraint c1
{ length<=10; length>7 ; }
```

```
int a
a = 8 + { $random } % 3
```

```
constraint c2
 { length inside { [8:10] } ; }
```

“verification engineers choose the best thing that fits into the VIP”

simplifying the code…

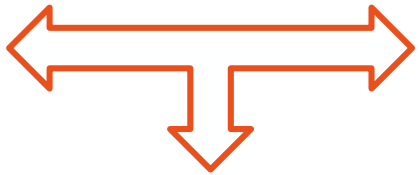
```
if ( obj . randomize )
   SUCCESSFUL, value=%d
else
      UNSUCCESSFUL
```

```
if  (  ! obj . randomize()  )
      $error ();
SUCCESSFUL, value=%d
```

“less number of codes provides more accessibility to VIP users”

assert( ) . . . simplifying the code

```
if ( obj . randomize() )
      PASS, a=%d;
else
      FAIL;
```



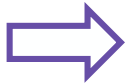
```
if ( obj . randomize() )
      PASS, a=%d;
else
      $error;
```

```
assert ( obj . randomize() ) ;
```

“number of times we use assert, makes the code more compact”

function return type . . . simplifying the code

`obj.randomize ()`



`void ' ( obj.randomize () )`

inserts warning (*w),
each time during a call

log file is clean,
no warnings

*W, TSNPK...

*W, MACNDF...

"engineers must use valid syntax, debug becomes easier"

rand vs randc . . . YES/NO ?

rand bit [2:0] a;

2 3 2 5 5 7 0 0

unequal distribution

randc bit [2:0] a;

5 7 3 0 2 1 6 4

equal distribution

“variables declared with rand keyword distribute values uniformly”

signed vs unsigned

```
int a1;
a1 = $urandom;
+N1 –N2 +Nn…
```

```
int unsigned a2;
a2 = $urandom;
+N1 +N2 +Nn…
```

```
int unsigned a3;
a3 = $random;
+N1 +N2 +Nn…
```

“signed variables handle with care !”

```
int b1;
b1 = $urandom_range(200,0);
+N1 +N2 +Nn…
```

implication and negation operator together . . .

```
rand  bit  [8:0]  datalength;
rand bit speed;
```

```
constraint c1 {
  if (!speed)
     datalength<=64;
  else
     datalength<=512;}
```

```
constraint c2 {
  (!speed) -> datalength<=64;}
```

“constraints are logical expressions,

helps to evaluate more number of seq. statement

SASKEN

Thank You :)