DAY-77
#100DAYSRTL

“System Verilog:-Constraints(Phase-2)”

“Introduction”:-

Constraints are a pivotal facet of randomization. Constraints serve as the avenue through which we articulate specific conditions or limitations on the randomness imbued in our values. Given the expansive nature of this subject, we've opted for a phased approach, unraveling the intricacies of constraints across multiple articles.

“Dist Constraints”:-

- Constraint provides control on randomization, from which the user can control the values on randomization.
- It would be good if it's possible to control the occurrence or repetition of the same value on randomization.
- It is possible, with a dist operator, that some values can be allocated more often to a random variable.
- This is also known as weighted distribution.
- dist is an operator, it takes a list of values and weights, separated by := or :/ operator

```
value := weight
value :/ weight
```

“:= Operator”:-

The “:= operator” assigns the specified weight to the item, or if the item is a range, specified weight to every value in the range.

```
addr dist { 2 := 5, [10:12] := 8 };

for addr == 2 , weight 5
    addr == 10, weight 8
    addr == 11, weight 8
    addr == 12, weight 8
```

“:/ Operator”:-

The “:/ operator” assigns the specified weight to the item, or if the item is a range, specified weight/n to every value in the range. where n is the number of values in the range.

```
addr dist { 2 :/ 5, [10:12] :/ 8 };

for addr == 2 , weight 5
    addr == 10, weight 8/3
    addr == 11, weight 8/3
    addr == 12, weight 8/3
```

“Code Practising”:-

```
class packet;
  rand bit [3:0] addr_1;
  rand bit [3:0] addr_2;
  constraint addr_1_range { addr_1 dist { 2 := 5, [10:12]:=9 }; }
  constraint addr_2_range { addr_2 dist { 2 :/ 5, [10:12]:/9 }; }
endclass:packet
module constr_dist;
 initial begin
   packet pkt;
   pkt = new();
   $display("Using := operator");
   repeat(10) begin
    pkt.randomize();
     $display("\taddr_1 = %0d" ,pkt.addr_1);
   end
   $display("-------------------------------------");
   $display("Using :/ operator");
   repeat(10)begin
    pkt.randomize();
     $display("\taddr_2 = %0d",pkt.addr_2);
   end
 end
endmodule
```

“Result”:-

```
Using := operator
      addr_1 = 11
      addr_1 = 11
      addr_1 = 12
      addr_1 = 11
      addr_1 = 2
      addr_1 = 12
      addr_1 = 10
      addr_1 = 2
      addr_1 = 10
      addr_1 = 2
-----------------------------------
Using :/ operator
      addr_2 = 12
      addr_2 = 11
      addr_2 = 12
      addr_2 = 2
      addr_2 = 11
      addr_2 = 10
      addr_2 = 11
      addr_2 = 2
      addr_2 = 2
      addr_2 = 10
```

“Implication if else Constraints”:-

The implication operator can be used to declare conditional relations between two variables.

- implication operator is denoted by the symbol ->.
- The implication operator is placed between the expression and constraint.
- If the expression on the LHS of implication operator (->) is true, then the only constraint on the RHS will be considered

“Code practising”:-

```
class packet;
  rand bit [3:0] addr;
  string    addr_range;
  constraint address_range { (addr_range == "small") -> (addr < 8);}
endclass
module constr_implication;
  initial begin
    packet pkt;
    pkt = new();
    pkt.addr_range = "small";
    $display("------------------------------------");
    repeat(4) begin
      pkt.randomize();
      $display("\taddr_range = %s addr = %0d",pkt.addr_range,pkt.addr);
    end
    $display("------------------------------------");
  end
endmodule
```

“Result”:-

```
------------------------------------
        addr_range = small addr = 1
        addr_range = small addr = 6
        addr_range = small addr = 0
        addr_range = small addr = 7
------------------------------------
```

“Code Practising”:-

```
class packet;
  rand bit [3:0] addr;
  string    addr_range;
  constraint address_range { if (addr_range == "small") (addr < 8);
                             else addr>8;}
endclass
module constr_implication;
  initial begin
    packet pkt;
    pkt = new();
    pkt.addr_range = "Big";
    $display("------------------------------------");
    repeat(4) begin
      pkt.randomize();
      $display("\taddr_range = %s addr = %0d",pkt.addr_range,pkt.addr);
    end
    $display("------------------------------------");
  end
endmodule
```

“Result”:-

```
-----------------------------------
        addr_range = Big addr = 11
        addr_range = Big addr = 10
        addr_range = Big addr = 11
        addr_range = Big addr = 9
-----------------------------------
```

“foreach loop Constraint Blocks”:-

- SystemVerilog supports using the foreach loop inside a constraint block.
- using the for each loop within the constraint block will make it easy to constrain an array.
- The foreach loop iterates over the elements of an array, so constraints with the foreach loop are called Iterative constraints

```
constraint constraint_name { foreach ( variable[iterator] ) variable[iterator]
<..conditions..>  }
```

“Code Practising”:-

```
class packet;
  rand byte addr [];
  rand byte data [];
  constraint avalues { foreach( addr[i] ) addr[i] inside {4,8,12,16}; }
  constraint dvalues { foreach( data[j] ) data[j] > 4 * j; }
  constraint asize   { addr.size < 4; }
  constraint dsize   { data.size == addr.size; }
endclass
module constr_iteration;
  initial begin
    packet pkt;
    pkt = new();
    $display("-------------------------------------");
    repeat(2) begin
      pkt.randomize();
      $display("\taddr-size = %0d data-size = %0d",pkt.addr.size(),pkt.data.size());
      foreach(pkt.addr[i]) $display("\taddr = %0d data = %0d",pkt.addr[i],pkt.data[i]);
      $display("-------------------------------------");
    end
  end
endmodule
```

“Result”:-

```
-----------------------------------
        addr-size = 3 data-size = 3
        addr = 16 data = 117
        addr = 12 data = 123
        addr = 4 data = 109
-----------------------------------
        addr-size = 0 data-size = 0
-----------------------------------
```

“Constraint modes”:-

- The **constraint_mode()** method can be used to disable any particular constraint block.
- By default, the constraint_mode value for all the constraint blocks will be 1.
- constraint_mode() can be used as follow,
  - ✓ addr_range.constraint_mode(0); //disable
  - ✓ packet.addr_range.constraint_mode(0);

“Static Constraints”:-

- A constraint block can be defined as static, by including a static keyword in its definition. static constraint addr_range { addr > 5; }
- Any mode change of static constraint will afect in all the objects of same class type

“Code Practising”:-

```
class packet;
  rand bit [3:0] addr;
  static constraint addr_range {addr > 5; }
endclass
module static_constr;
 initial begin
  packet pkt1;
  packet pkt2;
  pkt1 =new();
  pkt2 =new();
  pkt1.randomize();
  $display("\taddr = %0d",pkt1.addr);
  pkt1.addr_range.constraint_mode(0);
  $display("\tAfter disabling constraint");
  pkt1.randomize();
  $display("\taddr = %0d",pkt1.addr);
  pkt2.randomize();
  $display("\taddr = %0d",pkt2.addr);
end
endmodule
```

“Result”:-

```
addr = 9
After disabling constraint
addr = 2
addr = 11
```