CONSTRAINT

CODE EXAMPLE

JAIRAJ
DESIGN VER
ENGI

1. **Write an assertion to check clock period of 100 MHz.**

```
module test;
  // Clock period defined in nanoseconds
  time clk_period = 10.0 / 1.0ns;

  // Define a clock signal
  bit clk;

  // Clock generation: toggle clk every 5 ns to get a 10ns period (100 MHz
frequency)
  always #5 clk = ~clk;

  // Define a property to check the clock frequency
  property p1(int clk_period);
    realtime current_time;
    // Check if the clock period matches the expected value
    // Capture the current time at the rising edge of the clock
    // and ensure the difference between two rising edges matches clk_period
    (('1, current_time = $realtime) |=> (clk_period == $realtime -
current_time));
  endproperty

  // Assert the clock frequency at every positive edge of the clock
  clk_frequency: assert property (@(posedge clk) p1(clk_period))
    // If the assertion passes, display a message
    $display("%m  :: Time = [%0t]ns  clk frequency is correct, Assertion
Pass", $realtime);
  else
    // If the assertion fails, display an error message
    $error("%m :: Time = [%0t] clk not correct, Assertion fail", $realtime);

  // Main test sequence
  initial begin
    // Check the clock signal for a number of cycles
    for(int i = 0; i < 20; i++)
      @(posedge clk);
    // Finish simulation
    $finish;
  end

  // Initialize simulation dump for waveform analysis
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end
endmodule
```

2. Write an assertion to check clock frequency and duty cycle with +/- 5% error.

```
module test;

  logic clk = 0;

  property p1(int tolerance, realtime half_duty_cycle);
    realtime current_time;
    // At every positive edge of the clock, capture the current time
    @(posedge clk) (1, current_time = $realtime) |->
    // At the next negative edge, check if the high period is within the
specified tolerance
    @(negedge clk) ($realtime - current_time) >= half_duty_cycle -
(half_duty_cycle * tolerance) / 100.0 &&
                   ($realtime - current_time) <= half_duty_cycle +
(half_duty_cycle * tolerance) / 100.0;
  endproperty : p1

  property p2(int tolerance, realtime half_duty_cycle);
    realtime current_time;
    // At every negative edge of the clock, capture the current time
    @(negedge clk) (1, current_time = $time) |->
    // At the next positive edge, check if the low period is within the
specified tolerance
    @(posedge clk) ($realtime - current_time) >= half_duty_cycle -
(half_duty_cycle * tolerance) / 100.0 &&
                   ($realtime - current_time) <= half_duty_cycle +
(half_duty_cycle * tolerance) / 100.0;
  endproperty : p2

  // Assert the high period of the clock with a tolerance of 5% and a half-
duty cycle of 50ns
  clk_high: assert property (p1(5, 50ns));
  // Assert the low period of the clock with the same tolerance and half-duty
cycle
  clk_low: assert property (p2(5, 50ns));

  // Clock generation block
  always begin
    clk = 1'b1; // Set clock high
    #20; // Wait for 20 time units
    clk = 1'b0; // Set clock low
    #80; // Wait for 80 time units, creating a 100ns clock period with a
20/80 duty cycle
  end

  initial begin
    #2000 $finish;
  end

  // Setup for dumping waveforms to a .vcd file for analysis
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end

endmodule
```

3. Write a constraint to generate unique values in rows for a multidimensional array.

```
class packet;

  // Declare a 2D array 'a' of random bits with dimensions 4x4, each element
being 3 bits wide
  rand bit [2:0] a[4][4];

  // Define a constraint 'c1' on the elements of the 2D array 'a'
  constraint c1 {
    // Iterate over each element in the array using two nested foreach loops
    foreach (a[i,j])
      foreach (a[k,l])
        {
          // If the row indices i and k are different, ensure the elements in
the same column j are different
          i != k -> a[i][j] != a[k][j];

          // If the column indices j and l are different, ensure the elements
in the same row i are different
          j != l -> a[i][j] != a[i][l];
        }
  };

endclass

// Define a module named 'test'
module test;

  // Instantiate an object 'pkt' of class 'packet'
  packet pkt = new();

  initial begin
    // Randomize the object 'pkt' and display its contents
    pkt.randomize();
    $display("%p", pkt); // %p prints the variable in a format that shows
both the structure and the value
  end

endmodule
```

4. Write a constraint to generate diagonal values should be same in a multidimensional array.

```
class packet;

  // Declare a 2D array 'a' with dimensions 6x6, each element being 4 bits
wide
  rand bit[3:0] a[5:0][5:0];

  // Declare another randomizable 4-bit wide variable 'b'
  rand bit[3:0] b;

  // Define a constraint 'c1' to limit the values of 'b' to the range 0 to 9
  constraint c1 {b inside{[0:9]};}

  // Define a constraint 'c2' that applies to each element of the 2D array
'a',
  // ensuring all values are within the range 0 to 9
  constraint c2{foreach (a[i,j]){
      a[i][j] inside {[0:9]};}}

  // Define a constraint 'c3' that applies a special condition to the array
'a'
  // If an element is on the main diagonal (i==j) or the anti-diagonal (i+j
== 5),
  // it must be equal to 'b'
  constraint c3{foreach (a[i,j]){
        if(i==j || (i+j == 5)){
          a[i][j]==b;}}}

endclass

// Define a module named 'test'
module test;
  // Instantiate an object 'pkt' of class 'packet'
  packet pkt = new();

  initial begin
    // Randomize the object 'pkt' and then print the values of array 'a' and
variable 'b'
    pkt.randomize();

    // Iterate through the 2D array 'a' to display its elements
    foreach (pkt.a[i,j])
    begin
      $display("a[%0d][%0d] = %0d", i, j, pkt.a[i][j]);
    end

    // Display the entire array 'a' using the %p format specifier for
debugging
    $display("%p", pkt.a);

    // Display the value of 'b'
    $display("%d", pkt.b);
  end
endmodule
```

5. **Write a constraint to generate an address where 9 bits are always set to 1, and sequences of 111 or 000 should not occur in a 16-bit address.**

```
class packet;

  // Declare a 16-bit randomizable variable 'a'
  rand bit[15:0] a;

  // Define a constraint 'c1' on 'a'
  constraint c1 {
    // Ensure that the total number of 1's in 'a' equals 9
    $countones(a) == 9;

    // Further constraints applied to every triplet in 'a' to ensure
    // specific patterns are not formed
    foreach (a[i]) {
      // Check only up to the 14th bit to avoid out-of-bounds access
      if (i < 14) {
        // Ensure that no three consecutive bits form the pattern 000
        { a[i], a[i+1], a[i+2] } != 3'b000;
        // Ensure that no three consecutive bits form the pattern 111
        { a[i], a[i+1], a[i+2] } != 3'b111;
      }
    }
  }

endclass

// Define a module named 'test'
module test;

  // Instantiate an object 'pkt' of class 'packet'
  packet pkt = new();

  initial begin
    // Repeat the randomization and display process 10 times
    repeat(10) begin
      // Randomize 'pkt' according to its constraints
      pkt.randomize();
      // Display the 16-bit variable 'a' in binary format
      $display("%0b", pkt.a);
    end
  end
endmodule
```

6. Write a Constraint to generate a pattern 5,-10,15,-20,25.....

```
class packet;
  // Declare a randomizable array 'a' with 10 elements
  rand int a[10];
  // Declare another randomizable array 'b' with 9 elements
  rand int b[9];

  // Define a constraint 'c1' on arrays 'a' and 'b'
  constraint c1 {
    // Iterate over each element in array 'a'
    foreach (a[i]) {
      // Assign values to 'a' based on index parity
      if (i % 2 == 0)
        a[i] == i * -5; // If the index is even, assign the value as the
negative multiple of the index
      else
        a[i] == i * 5; // If the index is odd, assign the value as the
positive multiple of the index

      // Ensure that each element in 'b' is equal to the next element in 'a'
      b[i] == a[i + 1];
    }
  }
endclass

// Define a module named 'test'
module test;
  // Instantiate an object 'pkt' of class 'packet'
  packet pkt = new();

  initial begin
    // Randomize the object 'pkt'
    pkt.randomize();
    // Display the values of array 'b' after randomization
    $display("%p", pkt.b);
  end
endmodule
```

7. **Write a constraint to generate an address for different ports like port_0, port_1, port_2 and each Port is having starting and ending Address.**

```
class packet;
  // Define an enumeration type 'port_num' with three values: port_0, port_1,
port_2
  typedef enum {port_0, port_1, port_2} port_num;

  // Declare a randomizable variable 'port' of type 'port_num'
  rand port_num port;

  // Declare a randomizable 8-bit wide variable 'addr'
  rand bit [7:0] addr;

  // Define a constraint 'c1' on 'port' and 'addr'
  constraint c1 {
    // Constraints on 'addr' based on the value of 'port'
    port == port_0 -> addr inside {[0:10]};  // If 'port' is port_0, 'addr'
should be in the range [0:10]
    port == port_1 -> addr inside {[11:20]}; // If 'port' is port_1, 'addr'
should be in the range [11:20]
    port == port_2 -> addr inside {[21:30]}; // If 'port' is port_2, 'addr'
should be in the range [21:30]
  }
endclass

// Define a module named 'test'
module test;
  // Instantiate an object 'pkt' of class 'packet'
  packet pkt = new();

  initial begin
    // Repeat the randomization process 10 times
    repeat(10) begin
      // Randomize the object 'pkt'
      pkt.randomize();
      // Display the randomized values of 'port' and 'addr'
      $display("%p", pkt);
    end
  end
endmodule
```

8. Write a constraint to generate a pattern 9,99,999,9999,99999,..

```
class packet;

  rand int unsigned a;
  int q[$] = {0};

  // Declare integer variables 'var1' and 'var2'
  int var1;
  int var2;

  // Define a post_randomize function to be called after randomization
  // This function manipulates the variables and displays a post-
randomization value
  function void post_randomize();
    // Declare and initialize a local integer variable 'b'
    int b = 9;

    // Remove the last element from array 'q' and assign it to 'var1'
    var1 = q.pop_back();

    // Calculate 'var2' based on 'var1' and 'b'
    var2 = (var1 * 10) + b;

    // Insert 'var2' at the front of array 'q'
    q.push_front(var2);

    // Display the post-randomization value of 'var2'
    $display("POST_RANDOMIZATION::value=%d", var2);
  endfunction

  // Define a constraint 'c1' to ensure that 'a' is equal to the post-
randomization value 'var2'
  constraint c1 { a == var2; }

  // Define a display function to display the value of 'a'
  function void display();
    $display("Display value::a=%d", a);
  endfunction
endclass

// Define a module named 'test'
module test;
  initial begin
    // Create an instance 'pkt' of class 'packet'
    packet pkt = new();

    // Repeat the randomization process 9 times
    repeat(9) begin
      // Randomize the object 'pkt'
      pkt.randomize();

      // Call the display function to display the randomized value of 'a'
      pkt.display();
    end
  end
endmodule
```

9. Write a constraint to generate the Fibonacci series.

```
class packet;
  // Declare a dynamic array 'a' of unsigned integers
  rand int unsigned a[];

  // Define a constraint 'c1' to restrict the size of array 'a' to be within
the range [7:10]
  constraint c1 { a.size inside {[7:10]}; }

  // Define a post_randomize function to initialize the elements of array 'a'
after randomization
  function void post_randomize();
    // Initialize the first two elements of array 'a' with 0 and 1
    a[0] = 0;
    a[1] = 1;

    // Use a loop to calculate the Fibonacci sequence and assign values to
the remaining elements of 'a'
    for (int i = 2; i < a.size; i++)
      a[i] = a[i-1] + a[i-2];
  endfunction
endclass

// Define a module named 'test'
module test;
  // Declare an instance 'pkt' of class 'packet'
  packet pkt;

  // Instantiate the 'pkt' object
  initial begin
    pkt = new(); // Initialize the object 'pkt'

    // Randomize the object 'pkt'
    pkt.randomize();

    // Display the randomized values of array 'a' using %p format specifier
    $display("%p", pkt);
  end
endmodule
```

10. Write a constraint to generate palindrome.

```
class packet;
  // Declare a random 32-bit wide logic vector 'a'
  rand logic [31:0] a;

  // Define a post_randomize function to be executed after randomization
  function void post_randomize();
    // Iterate through the first half of the vector
    for (int i = 0; i < 16; i++) begin
      // Assign each element of the first half to its corresponding element
in the second half
      a[i] = a[31 - i];  // Palindrome logic: mirror the first half to the
second half
    end
  endfunction
endclass

// Define a module named 'test'
module test;
  // Declare an instance 'pkt' of class 'packet'
  packet pkt;

  // Initialize and randomize the object 'pkt'
  initial begin
    pkt = new();       // Instantiate the 'pkt' object
    pkt.randomize();   // Randomize the 'pkt' object

    // Display the randomized value of vector 'a'
    $display("Value of a: %b", pkt.a);
  end
endmodule
```

11. Write a constraint to generate Prime numbers in different ways.

```
class packet;
  // Declare a dynamic array 'a' of 8-bit width
  rand bit [7:0]  a[];

  // Declare an integer variable 'i' for loop iteration
  int i;

  // Define a constraint 'prime_numbers' to generate prime numbers and store
them in array 'a'
  constraint prime_numbers {
    // Constrain the size of array 'a' to be exactly 20 elements
    a.size == 20;

    // Iterate over each element 'a[i]' in the array using 'foreach' loop
    foreach (a[i])
      // Check if 'i' is a prime number
      if (!((i % 2 == 0 && i == 2) || (i % 3 == 0 && i != 3) || (i % 4 == 0
&& i != 4) || (i % 5 == 0 && i != 5) || (i % 6 == 0 && i != 6) || (i % 7 == 0
&& i != 7) || (i % 8 == 0 && i != 8) || (i % 9 == 0 && i != 9)))
        // If 'i' is a prime number, assign it to 'a[i]'
        a[i] == i;
      else
        // If 'i' is not a prime number, assign 1 to 'a[i]'
        a[i] == 1;
  }
endclass

// Define a module named 'test'
module test;
  // Declare an instance 'pkt' of class 'packet'
  packet pkt;

  // Instantiate the 'pkt' object
  initial begin
    // Initialize the 'pkt' object
    pkt = new();

    // Randomize the 'pkt' object, applying constraints defined in the
'packet' class
    pkt.randomize();

    // Display the randomized values of array 'a' using '%p' format specifier
    $display("pkt = %p", pkt.a);
  end
endmodule
```

////////////////////////////or////////////////////////////

```
class packet;

  rand bit[7:0] a[$], b[$];

  // Define a constraint 'c1' to ensure that the size of array 'a' is exactly
200 elements
  constraint c1 { a.size == 200; }

  // Define a constraint 'c2' to populate array 'a' with prime numbers
  constraint c2 {
    foreach (a[i])
      if (i > 1)
        // Constrain 'a[i]' to be a prime number using the 'prime_num'
function
        a[i] == prime_num(i);
  }

  // Function to check if a number is prime
  function int prime_num(int c);
    for (int i = 2; i < c; i++)
      if (c % i == 0)
        return 2; // Return 2 if the number is not prime (2 is a placeholder
indicating not prime)
    // If the loop completes without finding a divisor, 'c' is a prime number
    prime_num = c;
  endfunction

  // Function to be executed after randomization
  function void post_randomize();
    // Remove duplicate elements from array 'a'
    a = a.unique;

    // Iterate over the elements of 'a'
    for (int i = 0; i < a.size(); i++)
      // Check if the unit place of 'a[i]' is 7
      if (a[i] % 10 == 7)
        // If the unit place is 7, add 'a[i]' to array 'b'
        b.push_back(a[i]);
  endfunction
endclass

module test;

  packet pkt = new;

  initial begin
    void'(pkt.randomize());
    // Display the randomized values of array 'a' and 'b' using '%p' format
specifier
    $display("%p", pkt);
  end
endmodule
```

12. Write a constraint to randomly generate 10 unique prime numbers in an array between 1 and 200 & generated prime numbers should have 7 in unit Place (eg :7,17,37....)

```
class packet;

  rand bit[7:0] a[10];
  int b[10];

  // Define a function to calculate prime numbers and store them in array 'b'
  function void pre_randomize();
    int k = 0; // Initialize index for array 'b'

    // Loop to iterate through numbers from 7 to 200 with a step of 10
    for (int i = 7; i <= 200; i += 10) begin
      int c = 0; // Initialize a flag to check for prime

      // Loop to check if 'i' is a prime number
      for (int j = 2; j <= i/2 + 1; j++) begin
        // If 'i' is divisible by 'j', set the flag and break the loop
        if (i % j == 0) begin
          c = 1;
          break;
        end
      end

      // If 'c' is 0, 'i' is a prime number, so store it in array 'b'
      if (c == 0) begin
        b[k] = i;
        k++; // Increment the index for array 'b'
      end
    end
  endfunction

  // Define a constraint 'c1' to ensure that elements of 'a' match with prime
numbers from 'b'
  constraint c1 {
    foreach (a[i]) // Iterate over each element 'a[i]' in array 'a'
      a[i] == b[i]; // Constrain 'a[i]' to be equal to the corresponding
element in array 'b'
  }
endclass

// Define a module named 'test'
module test;
  // Declare an instance 'pkt' of class 'packet'
  packet pkt = new;

  // Initialize and randomize the object 'pkt'
  initial begin
    void'(pkt.randomize());

    // Display the randomized values of array 'a' using '%0p' format
specifier
    $display("%0p", pkt.a);
  end
endmodule
```

13. Write a constraint to implement randc construct using rand keyword.

```
class packet;
  // Declare a dynamic array 'a' of 11-bit width elements
  bit [10:0] a [$];

  // Declare a random 11-bit width variable 'b'
  rand bit [10:0] b;

  // Constraint 'c' ensures that 'b' is not inside 'a'
  constraint c { !(b inside {a}); }

  // Constraint 'c2' constrains the range of 'b' to be between 2 and 20
  constraint c2 { b inside {[2:20]}; }

  // Function to be executed after randomization
  function void post_randomize();
    // Add 'b' to the front of array 'a'
    a.push_front(b);

    // If the size of 'a' exceeds 10, remove the last element
    if (a.size() == 10)
      a.delete();
  endfunction
endclass

// Define a module named 'test'
module test;
  // Initialize and randomize the object 'pkt' and display its values
  initial begin
    // Create an instance 'pkt' of class 'packet'
    packet pkt = new;

    // Repeat the randomization process 10 times
    repeat (10) begin
      // Randomize the object 'pkt'
      void'(pkt.randomize());

      // Display the value of 'b' and the elements of 'a' for each iteration
      $display("b is %d %b ", pkt.b, pkt.b);

      // Iterate over each element of 'a' and display its value along with
its index
      foreach (pkt.a[i])
        $display("a[%0d] value is %d", i, pkt.a[i]);
    end
  end
endmodule
```

14. There is a memory which can take addresses 0x00 to 0x100 except the reserved memory from 0x20 to 0xE0. address should not take reserved memory. address should be 4 byte aligned memory . Write a constraint to generate 16 byte addresses. For example if it generates 0x0 now it should generate 0x4 ,next 0x8 next 0x12 addresses such that it generates 16byte addresses.

```
class packet;
  // Declare random variables 'addr' and 'cntr'
  rand bit [31:0] addr;
  rand bit [2:0] cntr;

  // Constraint 'c1': When 'cntr' is 0, constrain 'addr' to be within range
['h0:'h100]
  constraint c1 { cntr == 0 -> addr inside {['h0:'h100]}; }

  // Constraint 'c2': When 'cntr' is 0, ensure 'addr' is not within range
['h20:'hE0]
  constraint c2 { cntr == 0 -> !(addr inside {['h20:'hE0]}); }

  // Constraint 'c3': Constrain the lower 2 bits of 'addr' to be 0
  constraint c3 { addr[1:0] == 0; }

  // Constraint 'c4': When 'cntr' is not 0, 'addr' increments by 4
  constraint c4 { cntr != 0 -> addr == const'(addr + 4); }

  // Function to be executed after randomization
  function void post_randomize();
    // Reset 'cntr' to 0 if it was incremented to a non-zero value
    if (cntr++ == 0)
      cntr = 0;
  endfunction
endclass

// Define a module named 'test'
module test;
  // Declare an instance 'pkt' of class 'packet'
  packet pkt = new;

  // Initialize and randomize the object 'pkt' and display its 'addr' value
  initial begin
    // Repeat randomization process 10 times
    repeat (10) begin
      // Randomize the object 'pkt'
      pkt.randomize();

      // Display the value of 'addr' using '%0h' format specifier
      $display("addr = %0h", pkt.addr);
    end
  end
endmodule
```

15. Write a constraint to generate a pattern of 0011223344...

```
class packet;
  rand bit [3:0] a[];

  constraint c1 { a.size == 10; }
  constraint c2 {
    foreach (a[i])
      // If 'i' is even or odd, set 'a[i]' to half of 'i'
      if (i % 2 == 0 || i % 2 == 1)
        a[i] == i / 2;
  }
endclass

module test;

  packet pkt = new;

  initial begin
    // Randomize the object 'pkt'
    pkt.randomize();

    // Display the values of array 'a' using '%p' format specifier
    $display("a = %p", pkt.a);
  end
endmodule
```

16. Write a constraint to generate a pattern of 000111222333444555...

```
class packet;

  rand bit [3:0] a[];

  constraint c1 { a.size == 21; }

  constraint c2 {
    foreach (a[i])
      // If 'i' is even or odd, set 'a[i]' to one-third of 'i'
      if (i % 2 == 0 || i % 2 == 1)
        a[i] == i / 3;
  }
endclass

module test;

  packet pkt = new;

  initial begin
    pkt.randomize();

    // Display the values of array 'a' using '%p' format specifier
    $display("a = %p", pkt.a);
  end
endmodule
```

17. Write an SV constraint to make diagonal elements Zero.

```
class diagonal;
  // Declare a 3x3 array 'array' of 4-bit width elements
  rand reg [3:0] array[2:0][2:0];

  // Constraint 'c': Set diagonal elements to zero, and non-diagonal elements
to non-zero values
  constraint c {
    foreach (array[i,j]) {
      if (i == j) // Check if 'i' is equal to 'j' (diagonal element)
        array[i][j] == 0; // Set diagonal elements to zero
      else
        array[i][j] != 0; // Set non-diagonal elements to non-zero values
    }
  }
endclass

// Define a module named 'top'
module top;
  // Declare an instance 'd' of class 'diagonal'
  diagonal d;

  // Initialize and randomize the object 'd' and display its 'array' values
  initial begin
    // Create an instance of class 'diagonal'
    d = new();

    // Randomize the object 'd'
    d.randomize();

    // Display the randomized values of array 'array' using '%p' format
specifier
    $display("array = %p", d.array);
  end
endmodule
```