1. **A function that takes an integer array and number 'n' as argument and fills the array with first 'n' integers.**

**CODE:**

**class pack;**

**function int get\_array(ref int a[],n);**

**a = new[n];**

**foreach(a[i])**

**begin**

**a[i] = i+1;**

**end**

**return a[n];**

**endfunction**

**endclass**

**module top;**

**pack pk=new();**

**int array[];**

**int b=6;**

**initial begin**

**pk.get\_array(array,b);**

**$display("array: %0p",array);**

**end**

**endmodule**

**Output:**

**# KERNEL: array: 1 2 3 4 5 6**

**NOTE: in class-> automatic**

**In module -> static**

**So its fine if u don’t declare function as automatic**

**Link:** [**https://www.edaplayground.com/x/Fjxc**](https://www.edaplayground.com/x/Fjxc)

**https://www.edaplayground.com/x/Fjxc**

1. **A function that computes the remainder given two numbers 'a' and 'b' i.e. a mod b by repeated subtraction.**

**CODE:**

[**https://www.edaplayground.com/x/JZ\_R**](https://www.edaplayground.com/x/JZ_R)

**class pack;**

**function int get\_prime(int n);**

**bit flag =1'b0;**

**if (n == 0 || n == 1)**

**begin**

**flag = 1'b1;**

**end**

**for(int i=2; i< n/2 ;i++)**

**begin**

**if(n % i == 0)**

**begin**

**flag = 1'b1 ;**

**break;**

**end**

**end**

**if(flag == 1'b0)**

**$display("%0d is prime",n);**

**else**

**$display("%0d not prime",n);**

**endfunction**

**endclass**

**module top;**

**pack pk=new();**

**int b =2;**

**initial begin**

**pk.get\_prime(b);**

**// $display("array: %0p",array);**

**end**

**endmodule**

1. **A function that computes the remainder given two numbers 'a' and 'b' i.e. a mod b by repeated subtraction.**

**CODE:**

[**https://www.edaplayground.com/x/Shke**](https://www.edaplayground.com/x/Shke)

[**https://www.edaplayground.com/x/JZ\_R**](https://www.edaplayground.com/x/JZ_R)

[**https://www.edaplayground.com/x/JZ\_R**](https://www.edaplayground.com/x/JZ_R)

**class pack;**

**function int get\_prime(int a, b);**

**int rem;**

**if(b>a)**

**rem = a;**

**else**

**begin**

**do**

**begin**

**a = a-b;**

**rem =a;**

**end**

**while( b <= a) ;**

**end**

**return rem;**

**endfunction**

**endclass**

**module top;**

**pack pk=new();**

**int a = 14;**

**int b =3;**

**int z;**

**initial begin**

**z = pk.get\_prime(a,b);**

**$display("rem: %0d",z);**

**end**

**endmodule**

1. **A function which uses the fn. in 3 to fill an array with first 'n' primes.**

**Code:**

[**https://www.edaplayground.com/x/snnp**](https://www.edaplayground.com/x/snnp)

**class pack;**

**function int get\_prime(int n);**

**bit flag =1'b0;**

**if (n == 0 || n == 1)**

**begin**

**flag = 1'b1;**

**end**

**for(int i=2; i<= n/2 ;i++)**

**begin**

**if(n % i == 0)**

**begin**

**flag = 1'b1 ;**

**break;**

**end**

**end**

**if(flag == 1'b0)begin**

**// $display("%0d is prime",n);**

**return flag ;**

**end**

**else**

**begin**

**// $display("%0d not prime",n);**

**return flag;**

**end**

**endfunction**

**function int get\_array(int n);**

**int que[];**

**int x;**

**int m = 0;**

**que= new[n];**

**for(int j =0; j< (n\*n) ; j++)**

**begin**

**x= get\_prime(m);**

**if(x == 0)**

**que[j] = m;**

**m++;**

**end**

**$display("%0p",que);**

**endfunction**

**endclass**

**module top;**

**pack pk=new();**

**int b = 20;**

**initial begin**

**pk.get\_array(b);**

**// $display("array: %0p",array);**

**end**

**endmodule**

1. **A function to reverse a char array**

**Code using reverse method**

[**https://www.edaplayground.com/x/M6pQ**](https://www.edaplayground.com/x/M6pQ)

**code without reverse method:**

[**https://www.edaplayground.com/x/ZZxu**](https://www.edaplayground.com/x/ZZxu)

**//A function to reverse a char array**

**class char\_array;**

**function string reverse\_array( string a[]);**

**//reverse()-Return the reverse order of array element**

**a.reverse();**

**$display("reversed array = %p",a);**

**endfunction**

**endclass**

**module top;**

**char\_array ca = new();**

**string array[] = {"bangalore", "yelahanka", "maruthinagar", "oldtown", "newtown"};**

**initial begin**

**ca.reverse\_array(array);**

**end**

**endmodule**

**Other method:**

**class char\_array;**

**function string reverse\_array( string a);**

**int c= a.len();**

**int temp;**

**for(int i=0;i<c;i++)**

**begin**

**temp = a[i];**

**a[i]=a[c-1];**

**a[c-1] = temp;**

**c--;**

**end**

**$display("reversed array = %p",a);**

**endfunction**

**endclass**

**module top;**

**char\_array ca = new();**

**string array = "hello";**

**initial begin**

**ca.reverse\_array(array);**

**end**

**endmodule**

**CODE:**

**class char\_array;**

**function string reverse\_array( string a[]);**

**int c= a.size();**

**string temp;**

**for(int i=0;i<c;i++)**

**begin**

**temp = a[i];**

**a[i]=a[c-1];**

**a[c-1] = temp;**

**c--;**

**end**

**$display("reversed array = %p",a);**

**endfunction**

**endclass**

**module top;**

**char\_array ca = new();**

**string array[] = {"bangalore", "yelahanka", "maruthinagar", "oldtown", "newtown"};**

**initial begin**

**ca.reverse\_array(array);**

**end**

**endmodule**

1. **A function to compute the 1 bit CRC for an 31 bit integer by doing XOR of the 31 bits and fill it in the 32'nd bit**

**CODE:**

[**https://www.edaplayground.com/x/M7zb**](https://www.edaplayground.com/x/M7zb)

**class crc;**

**function int check\_xor(bit [31:0] a);**

**bit [30:0] b = a[30:0];**

**bit parity;**

**parity =^b;**

**$display("parity =%b",parity);**

**a [31] = parity ;**

**$display("a =%b",a);**

**$display("----------------------------------");**

**endfunction**

**endclass**

**module top;**

**crc c= new();**

**bit [31:0] d;**

**initial begin**

**repeat(10)begin**

**std::randomize(d) ;**

**$display(" d =%b", d);**

**c.check\_xor(d);**

**end**

**end**

**endmodule**

1. **Fork\_join**

**task sub\_run\_a();**

**while ($time < 50us) begin**

**#1us;**

**$display("sub\_run\_a(): ping at time %d", $time);**

**end**

**endtask : sub\_run\_a**

**// Task B**

**task sub\_run\_b();**

**#5us;**

**$display("sub\_run\_b() finished");**

**endtask : sub\_run\_b**

**// Task C**

**task sub\_run\_c();**

**#10us;**

**$display("sub\_run\_c() finished");**

**endtask : sub\_run\_c**

**fork**

**fork**

**sub\_run\_c();**

**sub\_run\_b();**

**join**

**sub\_run\_a();**

**join\_any**

**Link:**

[**https://www.edaplayground.com/x/8NqF**](https://www.edaplayground.com/x/8NqF)

# **Constraint**

1. **Write a constraint for the array[100] to generate a values in the range from 1 to 50. b)if i want 20 value in some random indexes how will you write the constraint ?**

**// Code your testbench here**

**// or browse Examples**

**class pack;**

**rand int a[100];**

**rand int n;**

**constraint c1 {foreach(a[i]) a[i]> 0 && a[i]< 51;}**

**constraint c2 { a[n] dist { 20:= 150, [1:50] := 20 };}**

**endclass**

**module top;**

**pack pk = new() ;**

**initial begin**

**for (int i =0;i<=10;i++)**

**begin**

**void'(pk.randomize()) ;**

**$display(" a [%0d] =%0p",i, pk.a) ;**

**end**

**end**

**endmodule**

1. **Write constraint for odd number Address generator and at every 5th iteration of the odd address the value should make to 0 and rest of all same odd number addresses only?**

**CODE:** [**https://www.edaplayground.com/x/PPAd**](https://www.edaplayground.com/x/PPAd)

**class address\_generator;**

**rand bit [5:0] odd\_address[25];**

**constraint c1{foreach(odd\_address[i]){**

**if((i+1)%5 == 4)**

**{**

**odd\_address[i] == 0 ;**

**}**

**else**

**odd\_address[i]%2!=0;**

**} }**

**// constraint c2 {foreach (odd\_address[i])**

**// odd\_address[] == 1'b1; }**

**endclass**

**module top;**

**address\_generator ag = new();**

**initial begin**

**repeat(6)**

**begin**

**void'(ag.randomize());**

**$display(" odd\_adress =%0p", ag.odd\_address);**

**end**

**end**

**endmodule**

1. **Fifo**

**class fifo #(parameter WIDTH, parameter DEPTH);**

**static bit [4:0] counter;**

**function void fifo\_read();**

**bit read\_enb =1;**

**counter--;**

**endfunction**

**function int fifo\_write();**

**bit write\_enb = 1;**

**bit [WIDTH-1:0] data\_in ;**

**std::randomize(data\_in) ;**

**counter++;**

**return data\_in;**

**endfunction**

**function int fifo\_full();**

**bit full;**

**if (counter == DEPTH)**

**full = 1;**

**return full;**

**endfunction**

**function int fifo\_empty();**

**bit empty ;**

**if (counter == 0)**

**empty =1;**

**return empty;**

**endfunction**

**endclass**

**module top;**

**fifo#(5,16) ff ;**

**int a;**

**initial begin**

**ff = new();**

**repeat(20)**

**begin**

**if(ff.fifo\_full() == 0)begin**

**a= ff.fifo\_write() ;**

**$display("a = %0d",a);**

**$display("counter = %0d", ff.counter);**

**end**

**else**

**$display(" fifo is full");**

**end**

**repeat(20)begin**

**if(ff.fifo\_empty() ==0)begin**

**ff.fifo\_read();**

**end**

**else begin**

**// $display("counter = %0d", ff.counter);**

**$display(" fifo is empty");**

**end**

**end**

**end**

**endmodule**

**CODE:**

[**https://www.edaplayground.com/x/G3GS**](https://www.edaplayground.com/x/G3GS)

1. **Fork\_join\_any**

**program main;**

**initial begin**

**#(10);**

**$display(" BEFORE fork time = %d ",$time );**

**fork**

**begin**

**# (20);**

**$display("time = %d # 20 ",$time );**

**end**

**begin**

**#(10);**

**$display("time = %d # 10 ",$time );**

**end**

**begin**

**#(5);**

**$display("time = %d # 5 ",$time );**

**end**

**join\_any**

**$display(" time = %d Outside the main fork ",$time );**

**#(40);**

**end**

**endprogram**

**Output:**

**# KERNEL: BEFORE fork time = 10**

**# KERNEL: time = 15 # 5**

**# KERNEL: time = 15 Outside the main fork**

**# KERNEL: time = 20 # 10**

**# KERNEL: time = 30 # 20**

1. **Fork\_join disable and disable fork**

**Code:**

[**https://www.edaplayground.com/x/pN3v**](https://www.edaplayground.com/x/pN3v)

**CODE: fork \_join\_none and fork\_join\_any disable**

[**https://www.edaplayground.com/x/FUHd**](https://www.edaplayground.com/x/FUHd)

**module disable\_fork;**

**initial begin**

**$display("----------------------------------------------------------");**

**//fork-1**

**fork**

**//Process-1**

**$display($time,"\tProcess-1 of fork-1 Started");**

**#5;**

**$display($time,"\tProcess-1 of fork-1 Finished");**

**//Process-2**

**begin**

**$display($time,"\tProcess-2 of fork-1 Started");**

**#20;**

**$display($time,"\tProcess-2 of fork-1 Finished");**

**end**

**join\_none**

**//fork-2**

**fork**

**//Process-1**

**begin**

**$display($time,"\tProcess-1 of fork-2 Started");**

**#5;**

**$display($time,"\tProcess-1 of fork-2 Finished");**

**end**

**//Process-2**

**begin**

**$display($time,"\tProcess-2 of fork-2 Started");**

**#20;**

**$display($time,"\tProcess-2 of fork-2 Finished");**

**end**

**join\_none**

**disable fork;**

**$display("---------------------------------------------------------");**

**$display($time,"\tAfter disable-fork");**

**$display("----------------------------------------------------------");**

**end**

**endmodule**

**Output:**

**# KERNEL: ----------------------------------------------------------  
# KERNEL: ---------------------------------------------------------  
# KERNEL: 0 After disable-fork  
# KERNEL: ----------------------------------------------------------**

1. **Constraint to generate two adjacent bits to be 1 and remaining bits to be 0 for bit [7:0]a;**

**CODE:** [**https://www.edaplayground.com/x/MQrt**](https://www.edaplayground.com/x/MQrt)

**//Constraint to generate two adjacent bits to be 1 and remaining bits to be 0 for bit [7:0]a;**

**class adj;**

**rand bit [7:0]a;**

**int count;**

**constraint c1{foreach(a[i])**

**if(a[i] == 1 && count <1 )**

**{**

**a[i+1] == 1;**

**i==i+2;**

**count == 1;**

**}**

**else**

**a[i] == 0;**

**}**

**constraint c2{ $countones(a) == 2 ;}**

**endclass**

**module top;**

**adj aj= new();**

**initial begin**

**repeat(5)begin**

**aj.randomize();**

**$display("a = %b", aj.a);**

**end**

**end**

**endmodule**

**14. Write a constraint to get 3 random addresses which should be 32 bits apart**

**CODE:**

[**https://www.edaplayground.com/x/8hbR**](https://www.edaplayground.com/x/8hbR)

**//Write a constraint to get 3 random addresses which should be 32 bits apart**

**class apart;**

**rand bit [7:0] a[3] ;**

**constraint c1{foreach(a[i])**

**if(i<2)**

**a[i+1] == a[i] +32 ;**

**}**

**endclass**

**module count;**

**apart ap =new();**

**initial begin**

**repeat(3) begin**

**void'(ap.randomize());**

**$display("a = %p", ap.a);**

**end**

**end**

**endmodule**

1. **Fork join . disable fork**

**CODE:** [**https://www.edaplayground.com/x/nnT2**](https://www.edaplayground.com/x/nnT2)

**module top;**

**initial begin**

**fork**

**begin //thread1**

**$display("process 1 started ,time = %0d " ,$time);**

**#5ns;**

**$display("process 1 finished ,time = %0d " ,$time);**

**end**

**begin //thread 2**

**sub\_process();**

**end**

**join\_none**

**$display("ended");**

**disable fork;**

**end**

**task sub\_process();**

**$display("Process 2 started, time = %0d " ,$time);**

**#4ns;**

**$display("process 2 finished ,time = %0d " ,$time);**

**endtask**

**endmodule**

1. **Fork join\_none**

**CODE:** [**https://www.edaplayground.com/x/iUDw**](https://www.edaplayground.com/x/iUDw)

**module fork\_test;**

**initial begin**

**for (int j=0; j<3; j++)**

**begin**

**fork**

**$display(j);**

**join\_none**

**end**

**end**

**endmodule**

**Output:**

**3**

**3**

**3**

1. **Scoreboard fifo**

**Code:** [**https://www.edaplayground.com/x/d5S5**](https://www.edaplayground.com/x/d5S5)

1. fork\_join\_none

CODE:

<https://www.edaplayground.com/x/fqQS>

module fork\_join\_none;

initial begin

$display("-----------------------------------------------------------------");

fork

//Process-1

$display($time,"\tProcess- Started");

begin

$display($time,"\tProcess-1 Started");

#5;

$display($time,"\tProcess-1 Finished");

end

//Process-2

begin

$display($time,"\tProcess-2 Startedt");

#20;

$display($time,"\tProcess-2 Finished");

end

join\_none

$display($time,"\tOutside Fork-Join\_none");

$display("-----------------------------------------------------------------");

end

endmodule

OUTPUT:

# KERNEL: -----------------------------------------------------------------  
# KERNEL: 0 Outside Fork-Join\_none  
# KERNEL: -----------------------------------------------------------------  
# KERNEL: 0 Process- Started  
# KERNEL: 0 Process-1 Started  
# KERNEL: 0 Process-2 Startedt  
# KERNEL: 5 Process-1 Finished  
# KERNEL: 20 Process-2 Finished