

Sistem Verilog

Klase, randomizacija, pokrivenost

Klase

- karakteristike
- nasleđivanje
- Virtualne klase
- Polimorfizam
- Parameterizovane klase

Klase

- Klasa je korisnički definisan tip podataka
- SV omogućava objektno orijentisano programiranje sa klasama
- klase definišu podatke i metode (taskove/funkcije) koje se izvršavaju nad datim podacima
- Objekti klasa mogu biti dinamički kreirani/ izbrisani/ dodeljeni/ pristupani u simulaciji
- Rukovalac objektima (handle) omogućava mehanizam rukovanja podacima nalik pokazivaču
- Objektno orijentisane tehnike uključuju
 - Nasleđivanje
 - Polimorfizam
 - Kapsulacija (prikrivanje podataka)
 - Apstraktno modelovanje
- Podaci klase se nazivaju polja
- Sabrutine klase se nazivaju metode
- Podaci i sabrutine se nazivaju članovi klase
- Metode klase se izvršavaju nad poljima klase

klase

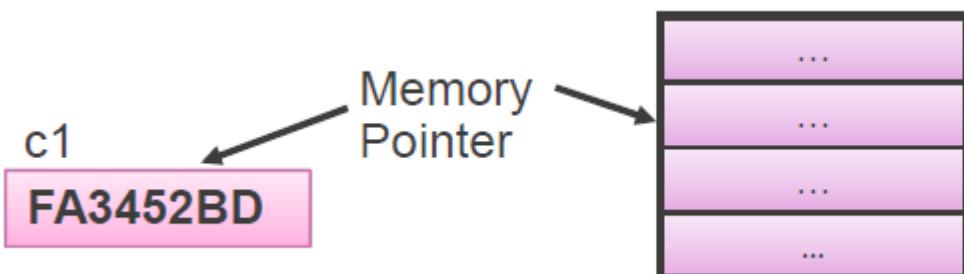
```
module class_mod;  
    ... ...  
    class user_class;  
        int data_var;  
        function int set;  
            return data_var ;  
        endfunction  
    endclass  
    ... ...  
endmodule
```

Instanca klase

- Klasa definiše tip podataka
- Slična je strukturi, ali je dinamički objekat
- Objekat je instanca klase
 - Deklaracija objekta : deklariše se kao varijabla (= null)
 - Kreiranje objekta: korišćenjem funkcije new
- Može se deklarisati i kreirati istovremeno
- Varijable sadrže rukovaće objektima
- Ne inicijalizovani rukovaoci objektima imaju vrednost null
 - Ne inicijalizovani objekti se mogu otkriti komparacijom sa null
- Objekt postoji ako je korišćen, a nije postavljen na vrednost null
- Funkcija new formira pokazivač na deo memorije

```
class myclass;  
    int data_var;  
endclass
```

```
myclass c1 ;  
c1 = new ;  
myclass c2 = new;
```



```
myclass c1;  
c1 = null ;
```

Pokazivači na rukovače klase

- SV automatski rukuje memorijom
- SV rukovaoc klase je sličniji C++
- SV automatski briše nekorišćenje objekte klase, prevenira curenje memorije, destruktor nije potreban
- Komparacija pokazivača i SV rukovaoca klase

Operacija	C pointer	C++ pointer	SV Handle
Aritmetičke operacije	dozvoljeno	dozvoljeno	nije dozvoljeno
Za proizvoljni tip podatka	dozvoljeno	dozvoljeno	nije dozvoljeno
Pokazivač na adresu	dozvoljeno	dozvoljeno	nije dozvoljeno
Dereferencing	dozvoljeno	dozvoljeno	nije dozvoljeno
Kastovanje	dozvoljeno	dozvoljeno	Ograničeno
Podrazumevana vrednost	undefined	0	null
Rukovanje memorijom	ručno	ručno	Automatski

Polja i metode klase

- Deklaracija klase definiše njene članice
 - Polja klase (podaci variable)
 - Metode klase (taskovi / funkcije)
- Kao kod struktura, članicama klase može se pristupiti sa “.”
- Poljima klase se pristupa direktno ili putem metoda klase

Polja i metode klase

```
module class_module;  
  class user_class;  
    int data_var;  
  
    function int get;  
      return data_var ;  
    endfunction  
  
    task set (int temp);  
      data_var = temp ;  
    endtask  
  endclass  
endmodule
```

```
user_class c1; ← declaring a class object  
c1 = new; ← creating a class object  
... ...  
c1.data_var = 22; ← Assigning a class property  
c1.set(44); ← directly  
$display(c1.get());  
... ...  
... ...
```

Retrieving a class property through the class method(function)

Deklaracija metoda

- Metode klase mogu biti unutar ili izvan klase
- Ključna reč `extern` opredeljuje da je ta metoda deklarisana van deklaracije klase.
- `class_name::method_name,`
:: scope resolution operator

Eksterna deklaracija metode primer

```
class my_class;  
    int count ;  
    extern task set (input int temp) ;  
    extern function int get ;  
endclass  
  
function int my_class::get ;  
    return count;  
endfunction  
  
task my_class::set (input int temp);  
    count = temp ;  
endtask
```

```
my_class obj = new;  
  
always @(posedge clk)  
begin  
    obj.set(44);  
    $display("display the value", obj.get());  
end
```

display the value 44

Konstruktori

- SV nema kompleksne tehnike alokacije/dealokacije memorije kao C++
- Nema curenje memorije, sakupljanje smeća (garbage collection) je automatski kao u Javi
- Funkcija new je posebna metoda klase koja definiše konstruktor
- new funkcija kreira objekat date klase i inicijalizuje mu vrednost na nulu
- Ime klase i ime konstruktora nisu isti.
- new funkcija nema return tip
- new funkciju korisnik može da specificira da inicijalizuje polja klase

```
class my_class;  
    integer count ;  
endclass  
  
my_class c1 = new ;
```

// c1.count value gets
// initialized to ZERO,0

```
class my_class;  
    integer count ;  
    function new ();  
        count = 5 ;  
    endfunction  
endclass  
  
my_class c1 = new ;
```

// c1.count = 5

```
class my_class;  
    integer count ;  
    function new (input int temp);  
        count = temp ;  
    endfunction  
endclass  
  
my_class c1 = new (44) ;
```

// c1.count = 44

Statička polja klase

- Statička polja dele sve instance klase
- Uobičajeno, istanca klase ima svoju jedinstvenu kopiju polja
- Statičkim poljima klase može se pristupiti čak i ako klasa nije kreirana ili je izbrisana
- primer:
 - Kreirani su novi objekti p1 i p2 iste klase
 - Objekat p3 je samo deklarisan nije kreiran
 - Svaki put kad je new funkcija pozivana, statičko polje zadržava prethodnu vrednost
 - Ostala polja klase imaju podrazumevane inicijalne vrednosti
 - Objekt p3 deli statičko polje, čak i ako nije kreiran

```
class sta_class;  
    static int count;  
    int temp;  
    function new;  
        temp = ++count;  
    endfunction  
endclass
```

```
sta_class p1 = new ;  
sta_class p2 = new ;  
sta_class p3 ;
```

```
initial begin  
    $display ( p1.temp, p1.count );  
    $display ( p2.temp, p2.count );  
    $display ( p3.count );  
end
```

1	2
2	2
	2

Statičke metode klase

- Statičke metode rade samo na statičkim poljima (varijablama)
- Statičke članice su pristupačne čak i ako je instanca klase izbrisana ili nije ni kreirana
- Može biti pozvana van klase čak i ukoliko ne postoji instance te klase ::
 - `class_name::static_function`
 - `class_instance.static_function`
 - Pokušaj pristupa ne statičkoj članici daće grešku
- `static task name (); ... endtask`
 - Može biti statički task sa automatskim varijablama

```
class sta_class;  
    static int count;  
    int temp;  
  
    function new;  
        temp = ++count;  
    endfunction  
  
    static function int getc();  
        return count ;  
    endfunction  
endclass
```

```
sta_class p1 = new ;
```

```
initial begin  
    $display (sta_class::getc());  
    $display (p1.getc());  
end
```

```
static function int getc();  
    return temp ;  
endfunction
```

1
1

Referenciranje na polje ili metod date instance THIS

- Referencira na polja ili metode date instance klase
- Ključna reč **this** je praktično rukovalac tog objekta klase
- **this** ključna reč se koristi za ne statičke metode klase
- Koristi se da referenciara identifikatore unutar klase lokalni doseg (local scope)
- Na sledećem primeru, ključna reč **this** se koristi da diferencira varijable deklarisane unutra i spolja.
- **this** referencira variable deklarisane lokalno unutar klase
- Vrlo korisno u slučaju hijerarhijski deklarisanih klasa
 - *Klase prethodnici i naslednici*

this primer

```
integer aa = 88 ;  
  
class demo;  
    integer aa;  
    extern task set (integer aa);  
    extern function int get ;  
endclass  
  
task demo::set (integer aa);  
    this.aa = aa ;  
endtask  
  
function int demo::get ;  
    return this.aa ;  
endfunction  
  
demo c3 = new;  
  
initial begin  
    $display ("Before",aa);  
    c3.set(77);  
    $display ("After",c3.get(),aa);  
end
```

Before 88
After 77 88

Before 88
After 0 88

Dodele, promene imena, kopiranje

```
class_name c1 ;  
initial begin  
    c1 = new;  
end
```

Create class **c1** (and initialize to default values)

```
class_name c1, c2 ;  
initial begin  
    c1 = new ;  
    c2 = c1 ;  
end
```

c2 class is created and **c1 class** copied to it

```
class_name c1, c2 ;  
initial begin  
    c1 = new ;  
    c2 = new c1 ;  
end
```

Create a **class c2** and copy the **class c1** to it.
Shallow copy

```
function trump;  
    class_name p5, p6 ;  
    p5 = new ;  
    p6 = new p5 ;  
endfunction
```

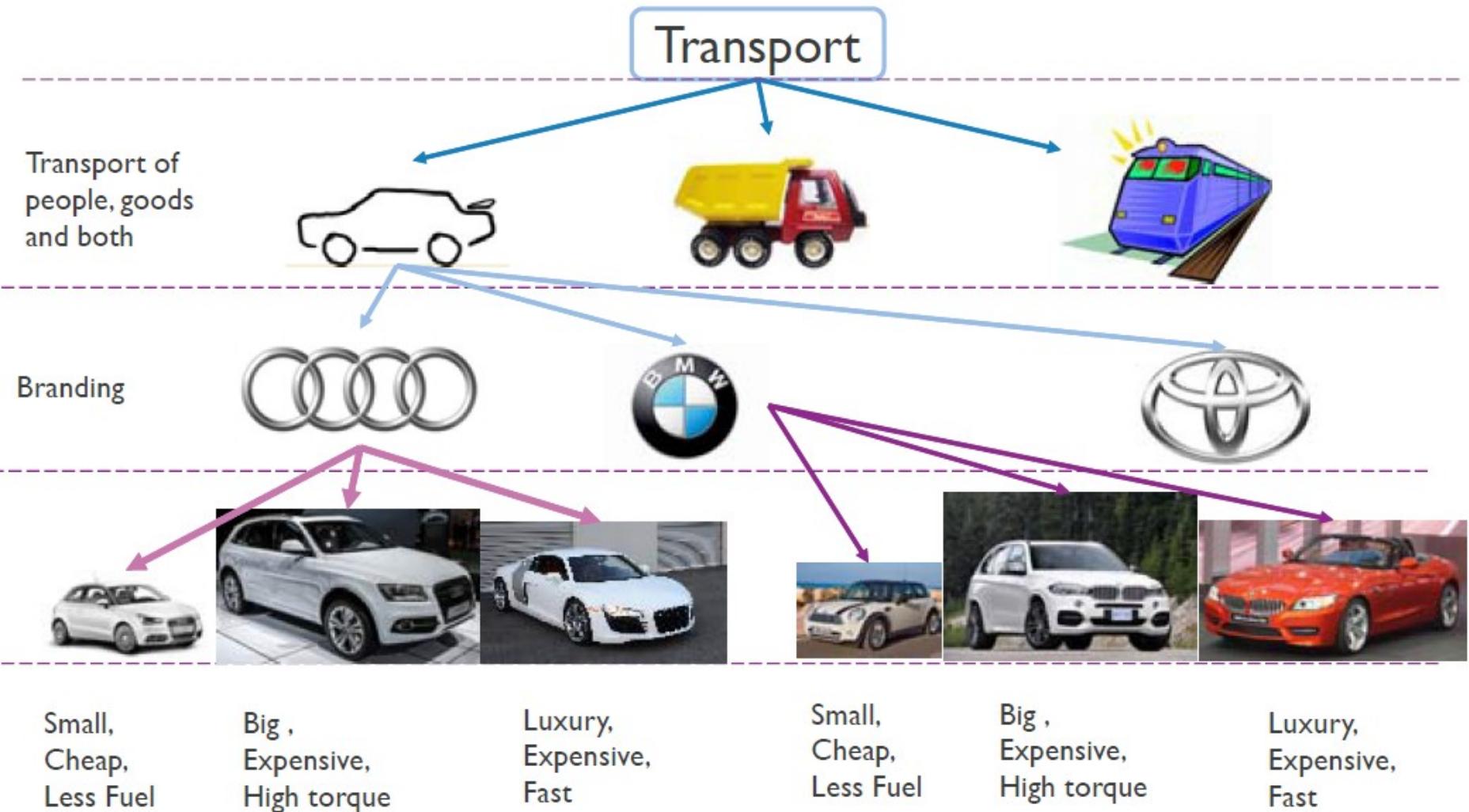
Classes can be declared, created, instantiated inside anywhere in the code

Classes declared, created and copied in a function

nasleđivanje

- OOP definicija
 - Nasleđivanje podrazumeva način izvođenja nove klase korišćenjem postojeće klase.
 - Nasleđivanje pomaže u ponovnom korišćenju postojećeg koda (code reuse)
 - Izvedene klase nasleđuju atribute i ponašanje klasa prethodnika
 - **Base class / Parent class / Super class**
- Izvedena ili klasa naslednik nasleđuje sve članice prethodne (roditeljske) klase
- SV dozvoljava jednostruko nasleđivanje (može se naslediti od samo jedne prethodne klase)
- Izvedena klasa može da doda nove članice ili da prepiše članice prethodne klase.
- Izvedena klasa sadrži sve članice prethodne klase, dok obrnuto ne važi.

Ilustracija nasleđivanja



- Izvedena klasa dobija sve članice prethodne klase
 - Izvedena klasa može da doda nove članice
 - Izvedena klasa može da prepiše članice prethodne klase
- Članice prethodne klase mogu se pozvati samo ukoliko su one i članice izvedene klase.
- Direktno se pozivaju operatorom “.”
- Ukoliko prethodna i izvedena klasa imaju definisanu funkciju new (konstruktor), prethodna se inicijalizuje kroz kroz new funkciju izvedene klase
- super.new;
- Pri pozivu new funkcije izvedne klase, super.new deklaracija unutar izvedene klase poziva prethodnu new funkciju.

Primer nasledivanja

```
class parent;  
int var1, var2;  
function new (input int add);  
    var1 = add ;  
    var2 = add * 2 ;  
    $display(var1);  
endfunction  
endclass
```

Parent class is re-useable

seperate child and parent
class is similar to this

```
class derived ;  
int var1, var2 ;  
int var3;  
function new (input int temp);  
    var1 = temp ;  
    var2 = temp * 2 ;  
    $display(var1);  
    $display(var2);  
    var3 = var2 * 2 ;  
    $display(var3);  
endfunction  
endclass
```

child/derived class inherits all
the class members from parent

```
class derived extends parent;  
int var3;  
function new (input int temp);  
    super.new (temp);  
    $display(var2);  
    var3 = var2 * 2 ;  
    $display(var3);  
endfunction  
endclass
```

Must be the first line of
the sub-class constructor

```
// derived c1 = new (5) ;  
initial begin  
    derived c1 ;  
    c1 = new (5) ;  
    c1.var1 = 6 ;  
    c1.var2 = 7;  
    c1.var3 = 8;  
end
```

results

5

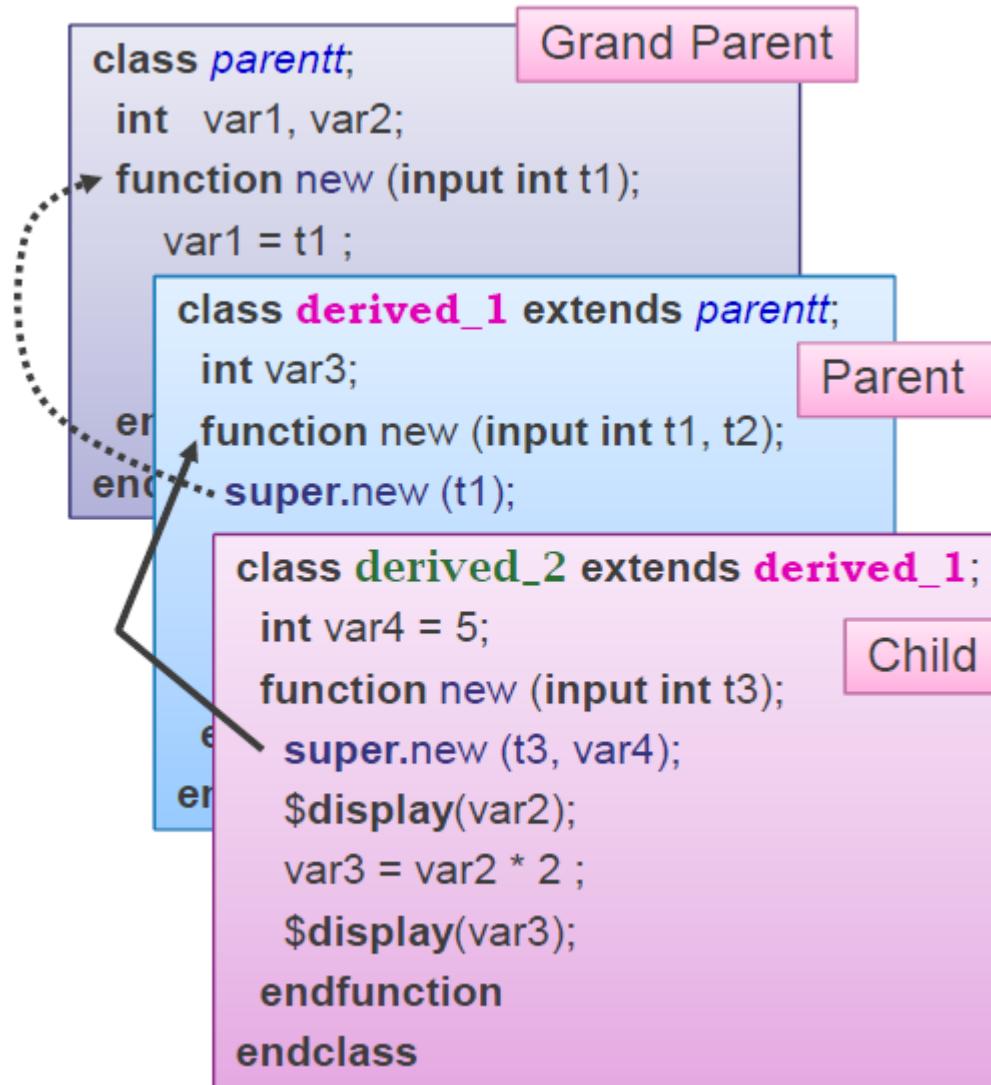
10

20

Slojevito nasleđivanje

- derived_1 klasa preuzima sve članice klase prethodnika parentt
- derived_2 klasa preuzima sve članice klase parentt i derived_1 klase.
- parentt klasa je prethodnik prethodnika klase derived_2
- super.new referencira direktnog prethodnika aktuelne izvedene klase
- Ne može se direktno pristupiti prethodniku prethodnika
- super.super.new nije dozvoljeno
- Super.parent_members je dozvoljeno (polja metode)

Slojevito nasleđivanje primer



Enkapsulacija/ prikrivanje podataka

- Podrazumevano je da se članicama klase može pristupiti spolja
- Postavaljanje restriktivnog pristupa članicama je osnovni OOP koncept
- local i protected
- local članicama se može pristupiti samo iz klase
- local članice nisu vidljive podklasama
- Po local članicama se može pisati / čitati iz klase, a ne iz podklase
- protected članicama se može pristupiti iz klase i iz podklase
 - Nisu vidljive / pristupačne spolja
- Mogu se nasleđivati sa prethodne na izvedenu klasu
- protected članice se nasleđuju, local članice se ne nasleđuju

local / protected primer

```
class parent;  
    local int var1; protected int var2;  
    task set (input int add);  
        var1 = add ; var2 = add * 2 ;  
    endtask  
    local function int get ();  
        get = var1;  
    endfunction
```

```
class newclass extends parent;  
    local int var3 ; int var4 ;  
    function int get();  
        //get = super.var1 ;// Error : Local  
        get = super.var2 ;  
        //get = super.get() ;// Error : Local  
    endfunction  
    task set (input int temp);  
        super.var2 = temp ;  
        //super.var1 = temp ;// Error : Local  
    endfunction
```

```
newclass p = new ;  
initial begin  
    //p.var1 = 44 ; // Error : Local  
    //p.var2 = 756; // Error: Protected  
    //p.var3 = 999; // Error : Local  
    p.var4 = 345; p.set(23);  
    $display ("Display Val ", p.get());end
```

Virtualna klasa

- Virtualna klasa i virtualne članice su prototipovi
- Mogu biti isključivo osnovna klasa iz koje se dalje izvode
- Nemože se instancirati
- Izvedene klase nasleđuju sve članice iz virtualne klase
- Mogu da sadrže virtualne metode
 - Virtualne metode ne moraju imati telo
 - Klase naslednici koji nasleđuju ovu virtualnu klasu moraju da definišu telo tim metodama
- Mogu da sadrže i ne virtualne metode
 - Tekve moraju biti definisane, moraju imati telo

Virtualna klasa primer

```
virtual class parent;  
    ....  
    virtual function get (input int val);  
    endfunction  
endclass
```

```
class add extends parent ;  
    ....  
    function get (input int val);  
        get = val + val ;  
    endfunction  
endclass
```

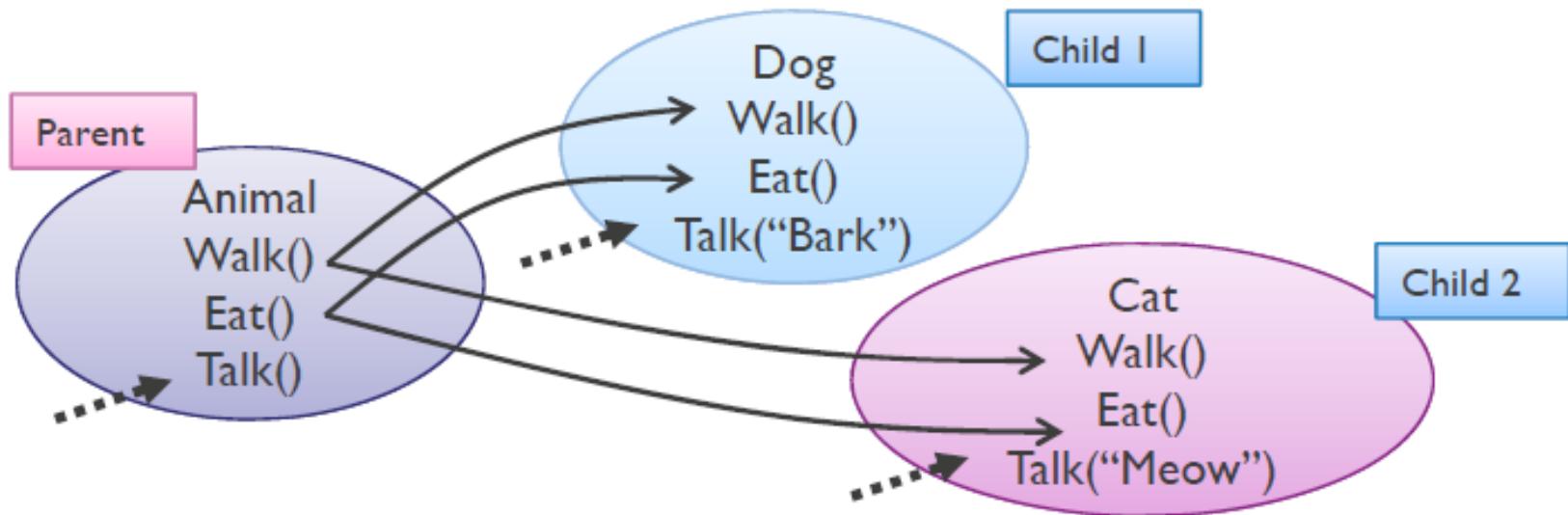
```
class mult extends parent ;  
    ....  
    function get (input int val);  
        get = val * val ;  
    endfunction  
endclass
```

```
add one = new ;  
mult two = new ;  
initial begin  
    $display (one.get(3), two.get(3));  
end
```

Polimorfizam

- “mogućnost pojave u više oblika”
- Sposobnost da se iste operacije izvedu na više različitih tipova stvari
- Objekt prethodne klase može da koristi zamenske metode kada barata sa objektima podtipa
- Zamenska metoda koju podklasa obezbeđuje mora imati isti potpis kao originalna metoda u super klasi.
- Za realizaciju polimorfizma identifikator 'virtual' se mora koristiti pri definisanju osnovne klase i metoda unutar nje.

Ilustracija polimorfizma



Polimorfizam

- Uniforman pristup različitim objektima
- Sposobnost da se u jednoj varijabli čuvaju objekti različitih tipova
- Ove osobine su korisne u fazama razvoja u kojima određeni detalji nisu poznati
- Simulator selektuje metode iz specifične klase tokom simulacije

Polimorfizam primer

- Šta ispisuje ovaj kod?

I am in Class C1
I am in Class C2
- Šta ukoliko klasa C2 nema funkciju M()?

I am in Class C1
I am in Class Poly
- Šta ukoliko u klasi Poly funkcija M() nije deklarisana kao virtualna?

I am in Class Poly
I am in Class Poly

```
class Poly ;  
    virtual function M();  
        $display("I am in Class Poly");  
    endfunction  
endclass
```

```
class C1 extends Poly;  
    function M();  
        $display ("I am in Class C1");  
    endfunction  
endclass
```

```
class C2 extends Poly;  
    function M();  
        $display ("I am in Class C2");  
    endfunction  
endclass
```

```
Poly pc [1:0]; C1 d1c = new; C2 d2c = new;  
initial begin  
    pc[0] = d1c ;  
    pc[1] = d2c ;  
    for (int i=0; i<2; i++) void'(pc[i].M());  
end
```

Apstraktna/ konkretna klasa

- Definicija: klasa C1 se naziva apstraktna ukoliko se koristi samo kao prethodna (superklasa) za druge klase
 - Klasa C1 samo specificira članice. Ne koristi se za kreiranje objekata
 - Izvedene klase moraju da definišu članice.
- Potreba za apstraktnim klasama se svodi na to da se može generalizovati da iz prethodne klase (super klase) naslednice koriste metode.
- Izvedena klasa iz apstraktne klase koja može da kreira objekte naziva se konkretna klasa
- Konkretna klasa se može instancirati, dok se iz apstraktne klase samo nasleđuje

Parametrizovane klase

- Parametrizovane klase koriste skup parametara da bi se kreirale potrebne specifične deklaracije
- Tipično se parametrizuje širina, veličina...
- Još se nazivaju Template klase ili generičke klase
- Parametrizovane klase se mogu proširivati

Parametrizovana klasa primer

```
module cp3;
    class class1 #(int size = 3);
        bit [size:0] box;
        task disp();
            $display(" Size of Box is %2d
                      ", $size(box));
        endtask
    endclass

    initial begin
        class1 b1 = new();
        class1 #(4) b2 = new();
        class1 #(5) b3 = new();
        b1.disp();
        b2.disp();
        b3.disp();
    end
endmodule
```

Size of Box is 4
Size of Box is 5
Size of Box is 6

pregled

- Objekti: instance klase
- klasa : dinamički objekt
- struktura : nije dinamička
- SV nema destruktor
- Virtualna klasa definiše
 - metode, virtualne metode
- Ne Virtualne klase
- Nasleđivanje
- Polimorfizam
- Parametrizovane klase

RANDOM STIMULUS

- Ciljevi
- Razumeti SV Randomizaciju i kako je koristiti u procesu HW verifikacije
- Zašto, kako , šta randomizovati
- Ograničavanje randomizacije
- Random Variable
- Metode Randomizacije
- Rand System funkcije/pozivanje
- Random/Thread/Object Stability

Šta se randomizuje

- **Šta treba randomizovati ?**
- Stimulus dizajna
- Randomizacija podataka ne pomaže mnogo
- Randomizacija kontrolne logike ponajviše pomaže u otkrivanju bagova
 - Konfiguracije uređaja
 - Redosled sekvenci
 - Konfiguracija okruženja
 - Kašnjenja u okviru defisanih granica
 - Dinamičke konfiguracije

Randomizacija uz ograničenja

- SV podržava Constraint Random Verification (CRV)
- Ograničenja - constraints
 - Specificira podskup stimulusa u bloku ograničenja
- Randomizacija
 - Slučajne funkcije, sa ili bez ograničenja, uniformna distribucija, težinska distribucija, Pre/Post randomizacija, deklaracije slučajnih varijabli i ne ponavljajuće sekvene
- Dinamička ograničenja
 - inline ograničenja, zaštićena ograničenja, aktivacija, deaktivacija ograničenja, prepisivanje blokova ograničenja
- Random Stabilnost
 - Stabilnost threadova, stabilnost objekata, seeding (zahtevanje identične ponovljive pseudoslučajne sekvene)

Randomizacija uz ograničenje primer

```
class mem ;  
    rand bit [10:0] data;  
    constraint c1 {data[5:0] =='b0;}  
endclass  
mem b = new ;  
initial begin  
    void'(b.randomize());  
    $display ("%b",b.data); // 10111000000  
end
```

```
t1 = $urandom(); // t1 is a 32 bit integer  
t2 = {$urandom,$urandom}; // t2 is a 64 bit integer  
t3 = $urandom_range(100,3); // randomize the values between the range
```

Slučajne varijable

- Varijable deklarisane sa ključnom reči rand jesu slučajne varijable
 - rand bit [3:0] data ;
- Varijable deklarisane sa ključnom reči randc jesu ciklične slučajen varijable
 - randc bit [1:0] sel
- Slučajno se prolazi kroz sve vrednosti u datom opsegu
- Ni jedna vrednost se ne ponavlja tokom ciklusa
- Ograničenja se mogu postaviti na te varijable
- Nizovi se mogu deklarisati sa rand ili randc
- Rand ili randc je dozvoljen samo unutar deklaracije klase

rand / randc primer

```
rand bit [3:0] data;  
// 4-bit meaning 16 possibilities;  
// same value repeating again with a  
probability of 1/16  
// Range 0 to 15 with equal probability
```

```
randc bit [1:0] sel;  
0 → 2 → 3 → 1 → // Cyclic order  
2 → 1 → 0 → 3 → // all possibilities  
2 → 0 → 1 → 3 → → // achieved
```

Blokovi ograničenja - constraint blocks

- Constraint block može da sadrži izraze ograničenja
- Ograničenje se može postaviti nad slučajnim varijablama
- Pokriva integralne tipove (bit, reg, logic, integer, enum, packed struct, etc.)
- Podržava vrednosti sa 2-stanja;
- Vrednosti sa 4-stanja ili operatori sa 4-stanja (== or !=) nisu dozvoljeni (X, Z) su zabranjeni
- Ograničenja se mogu omogućiti i onemogućiti
- Ograničenj ase mogu deklarisati van klase, ali unutar deklaracije klase

```
class test;  
  rand integer a, b ;  
  constraint c1 ;  
endclass  
  
constraint test::c1 {a < b;}
```

Note: No extern keyword
like function/task

Look at the
semi-colon

Nasleđivanje ograničenja

- Ograničenja prate ista pravila nasleđivanja kao polja i metode u klasama
- Ograničenja unutar izvedenih klasa ukoliko koriste isto ime prepisuju ograničenje prethodne klase
- Task randomize() je virtualan
- Constraint ima deklaraciju i telo sa {}
- Constraint se može deklarisati samo unutar klase

Ograničenja nasleđivanje primer

```
class A ;  
    rand integer comp ;  
    constraint c1;  
endclass  
  
constraint A::c1 { comp < 0 ;}  
  
class B extends A;  
    constraint c1 { comp > 0 ;}  
endclass
```

c1 constraint declared in class A ; constraint body defined outside class A

c1 constraint in class A is overridden by class B constraint

primer

```
class A;  
  rand logic [2:0] a, b, c ;  
  constraint c1 ;  
endclass  
constraint A::c1 {a < b;}  
  
class B extends A;  
  constraint c1 {a > b ;}  
endclass  
  
B bus = new;  
  
initial begin  
  repeat (10) begin  
    void'(bus.randomize());  
    $display (bus.a, ,bus.b, ,bus.c);  
  end  
end
```

a	b	c
2	1	1
1	0	1
7	3	2
6	3	3
5	3	2
4	0	4
3	2	2
5	4	1
2	1	7
7	4	1

Inferences

- Column a values are always greater than b ($a > b$)
- Value c is not constrained
- c values not related to a,b
- constraint c1's body is defined outside class A
- constraint c1 from class A is overridden by classB::c1
- randomize function call'd for the class instance **bus**, initiates the randomization
- if constraint randomization fails/not-satisfied , tool reports ERROR

inside

- Ograničenje koje podržava integer vrednosti (jedinične ili vektore)
- Inside operator specificira vrednost koja leži unutar opsega
- Negirana forma je !(expression inside {})

```
rand logic [2:0] a,b,c ;  
constraint c1 { (a inside { b , c }) ;} // a=5, b=5, c=6 satisfies the constraint
```

dist

- Ograničenje podržava težinske vrednosti zvane distribucije
- dist operator daje rezultat true, ukoliko je vrednost izraza unutar skupa, u ostalim slučajevima je false
- dist listi su dodeljene vrednosti (:=) razdvojene zarezima (,); Ukoliko je reč o opsegu tada vrednost navedena predstavlja težinu svakog entiteta u listi!
- “:/” operator dodeljuje težinu vrednosti, ili celom opsegu ukoliko je reč o opsegu. Ako opseg broji k vrednosti onda je težina svake od njih x/k

```
rand integer x ;
```

```
constraint c1 { x dist { 10 := 1 , 11 , [12:14] := 3, 17:= 6 } ;} //1,1,3,3,3,6
```

```
constraint c2 { x dist { [10:12] :/ 1 , 13 , 14 := 3, 17:= 6 } ;} //1/3,1/3,1/3,1,3,6
```

implikacija

- U okviru ograničenja (constraints) omogućene su dve konstrukcije uslovnih relacija
 - if ... else
 - Implikacija (->)
- if ... else: if (izraz) constraint_set1 (else) constraint_set2
- Implikacija (->): izraz -> constraint_set

Implikacija primer

```
class C ;  
rand logic [2:0] a ;  
rand logic sel ;  
constraint c6 {  
sel == 0 -> a < 3 ;  
sel == 1 -> a > 3 ;  
}  
endclass
```

<u>sel</u>	<u>a</u>
1	6
0	0
0	1
1	6
1	4
1	5
0	0
1	4

```
class C ;  
rand logic [2:0] a ;  
rand logic sel ;  
constraint c6 {  
if (sel == 0)  
a < 3 ;  
else if (sel == 1)  
a > 3 ;  
}  
endclass
```

Iterativna ograničenja

- Iterativna ograničenja su korisna na nizovima
- Varijable u petlji i varijable u indeksima se mogu koristiti

```
class C ;  
  rand int arr [3:0] ;  
  constraint c6 {foreach (arr[i]) ( i <= arr[i]) -> arr[i] <= i ;}  
endclass
```

Ograničenja sa redosledom

- Simulator mora proveriti da su vrednosti slučajne varijable iz skupa kombinacija dozvoljenih vrednosti
- Ograničenje “order”-redosled definiše simulatoru da razreši “s” pre razrešavanja “d”
- Epilog je taj da će za “s” biti odabrana vrednost true sa 50% verovatnoće, zatim će biti odabрано “d” kao posledica vrednosti “s”.
- $d == 0$ će se desiti u 50% slučajevea, dok se $d != 0$ događa u preostalih 50% slučajeva.

Upotreba “order”-a primer

```
class B;  
    rand bit s;  
    rand bit [31:0] d;  
    constraint c { s -> d == 0; } // c says, s implies d equals zero  
    constraint order { solve s before d; } // this statement solves s before d  
endclass
```

NOTE THE SEMICOLON

Funkcije u okviru ograničenja

- Ponekad su logički uslovi previše kompleksni da se uklope u implikaciju ili if..else
- Funkcije su u tom slučaju povoljno rešenje
 - Ne mogu da sadrže izlaze ili ref argumente
 - Treba da budu automatik (ne treba da čuvaju informaciju)
 - Ne mogu da modifikuju ograničenja

Funkcije u ograničenjima primer

```
function int incr (int var1)
    return incr++;
endfunction
class test;
    rand int count, temp ;
    constraint c1 { count == incr(temp) ; } // C2 constraint has to be satisfied too !
    constraint c2 { temp inside {2, 4, 6}; } // temp is executed first due to dependency
endclass
```

Metode randomizacije

- `randomize()`
- `randomize` funkcija predstavlja virtualnu metodu
- Svaka klasa ima ugrađenu (nasleđenu) `randomize()` virtualnu metodu
 - `virtual function int randomize();`
- Generiše slučajne vrednosti za sve aktivne rand varijable unutar tog objekta
- `randomize()` metod vraća 1 ako je uspešan, nulu ukoliko nije

randomize() primer

```
class D ;  
  rand logic [2:0] var1,var2,var3 ;  
  constraint c8 { var3 == var1 + var2 ;}  
endclass  
  
D dus = new ;  
  
initial begin  
  repeat (10) begin  
    if (dus.randomize()) $display ("Randomization succeeded");  
    else $display ("Randomization Failed");  
  end
```

pre_randomize i post_randomize

- pre_randomize i post_randomize su ugrađene funkcije koje automatski poziva randomize() funkcija
 - pre_randomize poziva se pre nego što su objekti randomizovani
 - post_randomize poziva se nakon što su objekti randomizovani
- Ove ugrađene funkcije korisnik može da prepiše

pre_randomize i post_randomize primer

```
class A;  
    function void pre_randomize();  
        $display ("I am in Class A");  
    endfunction  
endclass
```

```
class D extends A ;  
    rand logic [2:0] var1,var2,var3 ;  
    constraint c8 { var3 == var1 + var2 ;}  
    function void pre_randomize( );  
        if (super) super.pre_randomize();  
        $display ("I am in Class D Pre Rand", var1);  
    endfunction  
    function void post_randomize (int var3);  
        $display ("I am in Class D Post Rand",var1);  
    endfunction  
endclass
```

output
I am in Class A
I am in Class D Pre Rand X
I am in Class D Post Rand 2
Randomization succeeded

```
D dus = new ;  
initial begin  
    if (dus.randomize())  
        $display ("Randomization succeeded");  
    else $display ("Randomization Failed");  
end
```

In-line ograničavanje

- Korisnik može opredeliti da randomizuje specifičnu varijablu klase
- Može se formirati in-line ograničenje
- `randomize() ... with{set ;}`
- Ograničenje se može specificirati/dodati bez modifikacije osnovne klase

In-line ograničavanje primer

```
class A;  
  rand int x ;  
endclass
```

```
class B;  
  int x, y;  
  task doit (A aaa , int x, int z);  
    int success ;  
    success = aaa.randomize() with {x < y + z;};  
    $display(success); // 1  
  endtask  
endclass
```

```
B b = new;  
A a = new;  
initial begin  
  b.doit (a,2,3); // call the task in B  
end
```

take a look at the semi-colon

1 // says the success of the randomization

Check variable visibility

rand_mode metode

- rand_mode() metode su korisne za omogućavanje/onemogućavanje/proveru statusa randomizacije
 - **task** object[.random_variable]::rand_mode();
 - **function** int object[.random_variable]::rand_mode();
- Funkcija proverava status randomizacije aktivna ili ne
- Task se koristi da aktivira, deaktivira randomizaciju nad celim objektom ili pojedinim varijablama
- Ako je random varijabla neaktivna, ponaša se kao redovna varijabla
- Sve random varijalbe su podrazumevano inicialno aktivne
- rand_mode metode su ugrađene i nemogu biti prepisane

rand_mode primer

X
X
733
0
1

```
class A;  
    rand integer x,y;  
endclass  
  
A c1 = new ; integer tt ;  
  
initial begin  
    c1.rand_mode(0);  
    $display(c1.x);  
    void'(c1.randomize());  
    $display(c1.x);  
    c1.rand_mode(1);  
    void'(c1.randomize());  
    $display(c1.x);  
    c1.y.rand_mode(0);  
    tt = c1.y.rand_mode();  
    $display(tt);  
    c1.y.rand_mode(1);  
    tt = c1.y.rand_mode();  
    $display(tt);  
end
```

random class properties

rand_mode(0) task to disable all class random properties in class A

rand_mode(0) task disabled all class random properties in class A;
call randomize() function to check

rand_mode(1) task enables all class random properties in class A;

y.rand_mode(0) task disables class random property 'y' in class A;

rand_mode() function check class rand variable y in class A;

y.rand_mode(1) task enables class rand property 'y' in class A;

constraint_mode

- Slično kao rand_mode, constraint_mode() metode služe za omogućavanje/onemogućavanje/ proveru statusa blokova sa ograničenjima
- constraint_mode() funkcija vraća status bloka ograničenja
- constraint_mode() task definiše status bloka ograničenja

constraint_mode primer

```
class A;  
  rand int x,y;  
  constraint x1 { (x != 0) && ( x > -20) && ( x <= 20) ;}  
endclass
```

```
A c1 = new ; integer tt ;  
initial begin  
  c1.x1.constraint_mode(0);  
  void'(c1.randomize());  
  $display(c1.x);  
  c1.x1.constraint_mode(1);  
  void'(c1.randomize());  
  $display(c1.x);  
  tt = c1.x1.constraint_mode();  
  $display(tt);      end
```

-1098
19
1

Doseg randomizacije

- Svaka integralna varijabla se može randomizovati
- Varijable predajemo funkciji randomize() kao argumente
- randomize() funkcija vraća 1 ukoliko su varijable iz argumenta randomizovane, u suprotnom 0
- Ograničenja se takođe mogu dodati
- [std::]randomize([variable list])[with {constraint list / expression}]

```
logic [3:0] data,addr;
```

```
logic ok;
```

```
initial begin
```

```
ok = randomize (data,addr) with {data != addr;};
```

```
if (ok) $display("Success",ok);
```

```
else $display("Fail",ok);
```

```
end
```

Success 1

In-line random varijable

- `randomize()` metod bez argumenata randomizuje sve random varijable unutar objekta
- `randomize()` sa argumentima (kao argument može biti i ne random varijabla)
 - Tada će i ona biti randomizovana

in_line random primer

```
class A;  
rand byte x,y;  
byte u, v ;  
constraint x1 { (x != 0) && ( x > -20 + u ) && ( x <= 20 + v) ;}  
endclass  
  
A aa = new ; integer tt ;  
  
initial begin  
    void'(aa.randomize()); $display (aa.x,,aa.y,,aa.u,,aa.v);  
    void'(aa.randomize(x)); $display (aa.x,,aa.y,,aa.u,,aa.v);  
    void'(aa.randomize(u,v)); $display (aa.x,,aa.y,,aa.u,,aa.v);  
    void'(aa.randomize(x,y)); $display (aa.x,,aa.y,,aa.u,,aa.v);  
    void'(aa.randomize(u,x)); $display (aa.x,,aa.y,,aa.u,,aa.v);  
end
```

Inferences
?

x	y	u	v
8	-72	0	0
19	-72	0	0
19	-72	-16	13
-13	65	-16	13
-64	65	-96	13

Slučajni broj (funkcije i metode)

- `$random` // vraća 32bitni pseudoslučajni neoznačeni integer, može mu se kao argument proslediti seed vrednost, radi ponovljivosti sekvene
- `$random_range(int unsigned maxval, int unsigned minval = 0)` // vraća pseudoslučajnu vrednost u definisanom opsegu

urandom i urandom_range primeri

```
int value;  
bit [63:0] addr;  
  
initial begin  
    value = $urandom();                      // 5422  
    value = $urandom(254);                    // 20488 (using the seed 254)  
    value = $urandom_range(22,33);            // 26  
    value = $urandom(254);                    // 20488 (regenerate data with same seed 254)  
    addr = {$urandom(),$urandom()};          // 17700061744047726739  
end
```

randcase sa dodatim težinskim faktorima

- randcase je case izraz koji slučajno selektuje jednu od svojih grana
- Težinski faktori se postavljaju uz svaku granu s leva
- Težinski faktori mogu biti izrazi
 - Izrazi ne mogu biti negativni

```
randcase
 1 : count = 1 ;
 4 : count = 5 ;
 3 : count = 19 ;
 2 : count = 32 ;
endcase
```

```
randcase
  x :      count = 1 ;
  x + y :   count = 5 ;
  x + y + z : count = 19 ;
  z - y :   count = 32 ;
endcase
```

Weights are assigned on left ;
selection sequence is random
Weights : $1 + 4 + 3 + 2 = 10$
Probabilities are $1/10, 4/10, 3/10, 2/10$

randsequence – slučajna sekvenca

- Randsequence generiše slučajnu sekvencu izraza
 - Selekcija bazirana na “produkcionoj listi”
- Produkciona lista ima ime i produkcone elemente
 - Produkcioni element se klasificuje u terminale ili ne terminale
 - Terminali su ne deljivi elementi kojima ne treba dalji opis
 - Ne terminali su definisani drugim terminalima
- Celokupna produkciona lista se dekomponuje na terminale

randsequence – primer

```
randsequence ( main )
  main : a1 a2 done ;
  a1  : one | two ;
  a2  : two | done ;
  one : {$display (" one ");};
  two : {$display (" two ");};
  done : {$display (" done ");};
endsequence
```

Production valid possibilities

one two done
one done done
two two done
two done done

randsequence – sekvenca sa težinskim faktorima

- Produkcionoj listi mogu se dodati težinski faktori
- Podrazumevana težina je 1
- Koristi se operator “:=“ za dodelu težina
- Elementi produkciione liste se odvajaju simbolom “|”

```
int one_1,two_2,three_3;  
initial begin  
repeat(60)  
    randsequence( main )  
        main : one := 1 | two := 2 | three := 3;  
        one : {one_1++};  
        two : {two_2++};  
        three: {three_3++};  
    endsequence  
$display(one_1,,two_2,,three_3);
```

10 17 33

If..else produkciona lista

- Produciona lista može biti uslovljena kroz if..else izraz
- Dato uslovljavanje mora biti Boolean

```
int sel = 0 ;
int a,b,c;
repeat(10)
randsequence( entry )
    entry : one two ;
    one  : { if(sel) a++; else b++; };
    two  : { c++; };
endsequence
$display (a,,b,,c);
```

0	10	10
---	----	----

repeat produkciona lista

- Koristi se za ponavljanje elementa produkcije specificiran broj puta

```
int sel = 0 ;
int aa,bb,cc;
repeat(60)
randsequence (main)
    main : one | repeat (2) two | repeat (3) three ;
    one  : {aa++;};
    two  : {bb++;};
    three : {cc++;};
endsequence
$display (aa,,bb,,cc);
```

Napuštanje produkcije – break, return

- break i return izrazi se mogu koristiti za napuštanje produkcije
- break izraz terminira generisanje sekvence, dakle napušta se randsequence blok.
- return za razliku od break izraza samo prekida generisanje trenutne produkcije i nastavlja sledeću.

```
int flag = 2 ;
initial begin
    randsequence()
        all : S1 S2 ;
        S1 : A B C ;
        S2 : A { if( flag == 1 ) return; } B C ;
        A : { $write( "A" ); } ;
        B : { if( flag == 2 ) return/break; $write( "B" ); } ;
        C : { $write( "C" ); } ;
    endsequence
    $display(""); end
```

return used

ACAC

break used

A

random stability (stohastička stabilnost)

- U verilogu, ako se izvorni kod promeni, čak i ako se sačuva isti seed, simulator može da generiše različit stimulus
 - random(seed), podsećamo se da je seed ulaz u generator pseudoslučajnih brojeva RNG
- Kako SV ovo rešava
- Svaki thread i objekt ima svoj lokalni generator slučajnih brojeva
- Obzirom na ovu lokalnost, svaki thread/objekat je nezavisan po pitanju slučajne sekvence od drugih
- Ova karakteristika se naziva random stability

Random stability - karakteristike

- Stabilnost thread-a
 - Svaki thread ima nezavisan RNG za sve pozive funkcijama randomizacije
 - Svaki put pri kreiranju novog thread-a njegov RNG se inicira (seed-ing) sledećom slučajnom vrednošću njegovog thread-a prethodnika (hijerarhijsko iniciranje)
 - Stabilnost programa i thread-a je garantovana, sve dok se kreiranje thread-ova i generisanje slučajnih brojeva vrši istim redosledom kao i ranije.
 - Dodavanje novih thread-ova na postojeće testove, može se vršiti na kraju bloka koda da bi se očuvala random stabilnost prethodno kreirane simulacije

Random stability - karakteristike

- Stabilnost objekta
 - Svaka instanca klase (objekt) ima svoj nezavisni RNG za sve rand metode unutar njega
 - Pri kreiranju objekta korišćenjem new konstruktora, njegov RNG je iniciran (seeding) sledećom slučajnom vrednošću iz thread-a koji je kreirao objekat
 - Isto kao kod stabilnosti thread-a, stabilnost objekta je takođe garantovana sve dok su objekti i thread-ovi kreirani istim redosledom kao ranije.
 - U cilju očuvanja random stabilnosti potrebno je nove objekte i thread-ove kreirati tek nakon kreiranja postojećih.
- Ručna inicijalizacija (seeding)
 - Svi RNG mogu se ručno inicijalizovati (seeding)
 - U kombinaciji sa hijerarhijskom inicijalizacijom, moguće je u potpunosti obezbediti ponovljivu inicijalizaciju sa početnom inicijalizacijom samo jednog početnog thread-om iz kog su svi ostali izvedeni

Random stability primer

```
module stablity;
    class class_rand_seed;
        rand logic [3:0] Var;
        function new (int seed);
            srandom (seed);
            $display (" SEED is initialized to %2d ",seed);
        endfunction
        function void post_randomize();
            $display(" %2d ",Var);
        endfunction
    endclass
    class_rand_seed rs;
    initial begin
        rs = new(20);
        repeat(5) void'(rs.randomize());
        rs = new(1);
        repeat(5) void'(rs.randomize());
        rs = new(20);
        repeat(5) void'(rs.randomize());
    end
endmodule
```

SEED is initialized to 20

12
4
0
15
8

SEED is initialized to 1

15
7
3
14
14

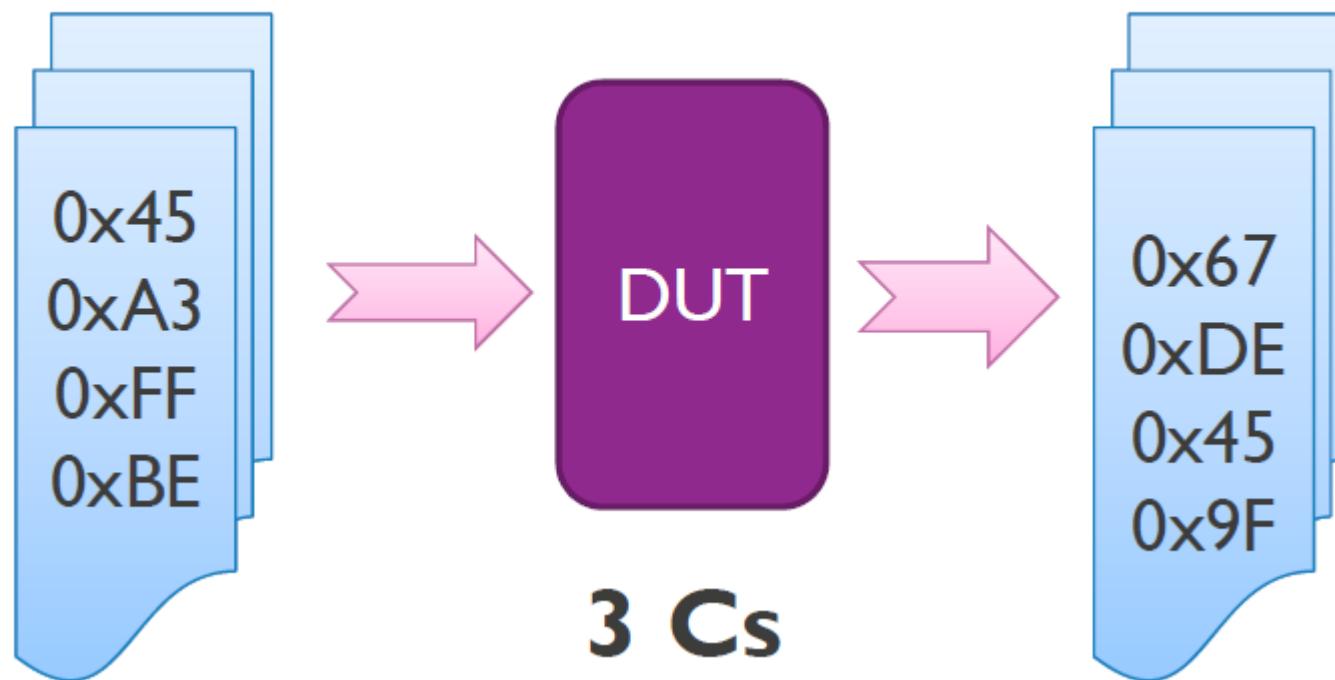
SEED is initialized to 20

12
4
0
15
8

Ograničena random verifikacija

- Šta predstavljaju 3 ?
 - Checker
 - Functional Coverage
 - Constraints

Constrained
Random stimulus



pregled

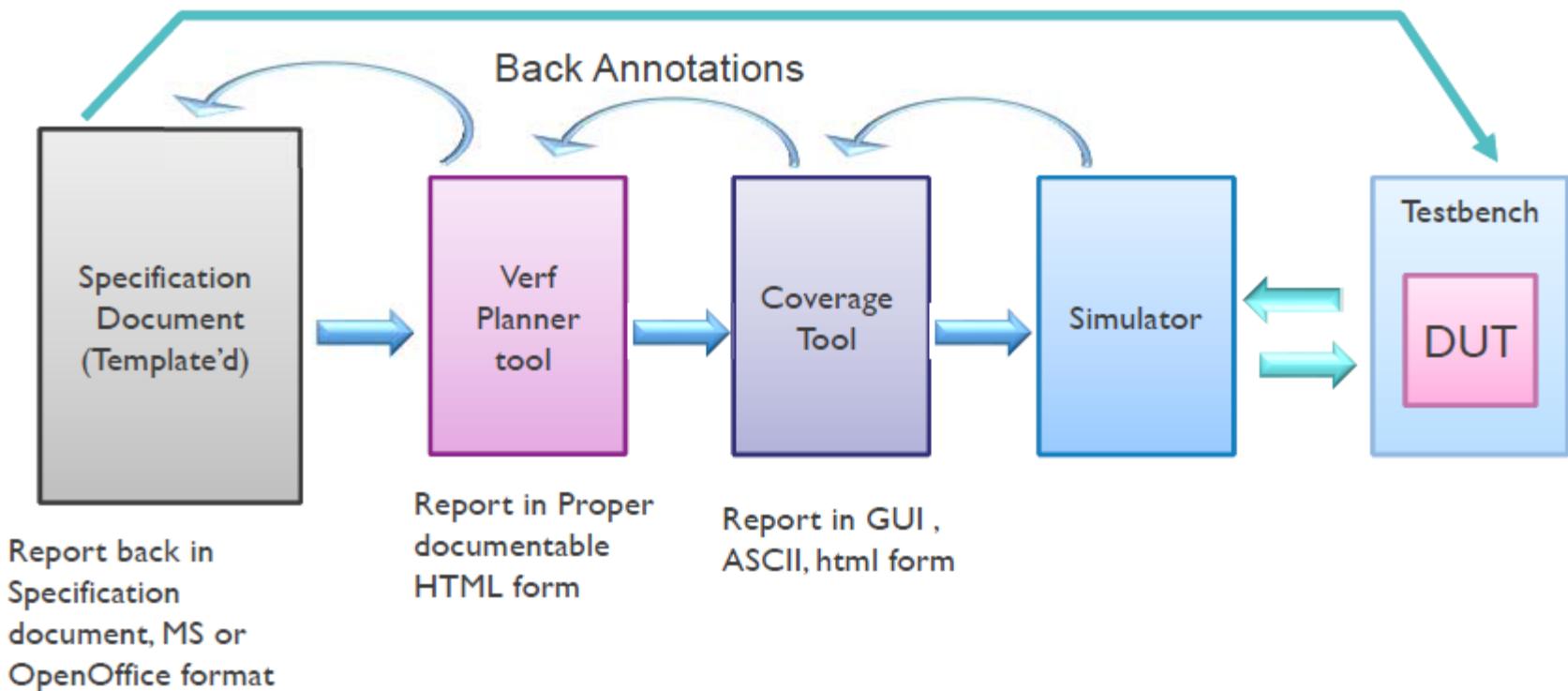
- Randomizacije pomaže u sledećem
 - Generisanju šireg stimulusa uz upotrebu kompaktnog koda
 - Kontrolu stimulusa
 - Pseudo slučajnost stimulusa
 - Bolje pokrivanje graničnih slučajeva
- Ograničavanje slučajnih varijabli
 - Kreiranje dozvoljenih vrednosti potrebnih u verификацији
 - Randomizacija modova/metoda
 - Random pozivi (`$urandom`, `$urandom_range`, `$random...`)
 - Random Thread/Objekt stabilnost

Coverage - pokrivenost

- Svrha
 - Razumeti mehanizam koji će nam reći do koje mera testirati. Koliko verifikacije je dovoljno?
- Pokrivenost
- Pokrivenost koda
- Funkcionalna pokrivenost
- Bins (korpe)
- Covergoup
- Coverpoint
- Coverage options/methods

Problemi verifikacije

- Verifikacija nije direktno kontrolisana, RE-SPIN !
- Koliko verifikacije je dovolno na projektu ?
- Kada zaustaviti verifikaciju ?
- Postoji li mera verifikacije?



pokrivenost

- Funkcionalna verifikacija je kompleksan zadatak pri verifikaciji sistema
- Pokrivenost jeste deo verifikacije, obezbeđuje potvrdu da je projektna specifikacija ispunjena
- Dve vrste pokrivenosti postoje u SV
 - Code / line / expression / Structural Coverage (uglavnom je zovemo pokrivenost koda)
 - Functional coverage (funkcionalna pokrivenost)
- Pokrivenost koda
 - *Automatski se izvodi iz RTL koda sistema*
 - *Simulacioni alati prate pokrivenost koda tokom simulacije*
- Funkcionalna pokrivenost
 - *Verifikacija korisnički definisanih scenarija*
 - *Bazira se na design specifikaciji, nezavisno od design koda ili njegove strukture*
 - *Dodaju se varijable koje pomažu u praćenju funkcionalne pokrivenosti*

Pokrivenost koda

- Simulator na zahtev prati pokrivenost koda i formira precizan izveštaj
- Podrazumevano je ova aktinost ne aktivna, jer intenzivno troši simulacione resurse

```
vlog -sv file_name.sv --cover {b c e s t f x}  
{branch code expr statement toggle fsm exclusions }  
vsim --coverage top
```

✓	6	bit clk;
	7	initial for(int i =0; i <5; i++) clk = #1 ~clk;
	8	
	9	
✓	10	initial begin
	11	x = 99;
	12	y <= x;
	13	#1 z = y;
	14	\$display ("zzzz", z, "at time", \$time);
X _B	15	if (x == 100)
X _S	16	\$display (" X is 100 ");
	17	

Funkcionalna pokrivenost

- SV obezbeđuje dve vrste funkcionalnog pokrivanja
 - Orientisano ka podacima - Data oriented (Covergroups)
 - Orientisano ka kontroli - Control oriented (SystemVerilog Assertions)
- Covergroup : može da uključi
 - kloking događaj za sinhronizaciju
 - skup cover points-a
 - Među pokrivenost (Cross coverage between cover points)
 - Opcine formalne argumente
 - Coverage options
 - Covergroup predstavlja SV klasu koja je nezavisna od vrste simulatora
- covergroup se instancira korišćenjem new() operatora

covergroup

```
enum logic [2:0] { HLT, SKZ, ADD, AND, XOR, LDA, STO, JMP} opcode;  
covergroup cg1 @(posedge clk);  
    c1: coverpoint opcode {           // c1 name assigned for opcode coverage  
        bins low = {HLT, SKZ };      // for each HLT and SKZ, low is incremented  
        bins mid = {ADD, AND, XOR }; // bins keep the count for coverage  
        bins high = {LDA, STO, JMP};  
    } cg1test : cross c1, c2 ;  
    c2: coverpoint addr;           // c2 name assigned with addr coverage  
endgroup
```

Funkcionalna pokrivenost

- Korisnik definiše
 - Testni plan kojim pokriva željenu funkcionalnost
 - Na ovnovu specifikacije dizajna, a ne na bazi implementacije (bitan detalj!)
- Simulator prati – meri kada su testovi ispunili tražene specifikacije
- Covergroup konstruktora može biti definisan u modulu, programu, interfejsu ili klasi
- kloking/sampling događaj specificira kada su cover points semplovani
- covergroup može takođe da specificira među pokrivenost (cross coverage) između coverpoints ili varijabli (sve kombinacije su moguće)

Covergroup – coverpoint - cross coverage primer

```
covergroup cg1 [@(event)] ;  
endgroup  
cg1 instance_name = new ;
```

```
enum { red, blue, green } color;  
bit [2:0] addr, offset, temp;  
covergroup g2 @(posedge clk);  
    c1: coverpoint temp;  
    c2: coverpoint offset;  
    aXb : cross color, addr; // cross coverage ..variables  
    all : cross color, c1, c2; // between variables & coverpoint  
endgroup
```

Automatske korpe (automatic bins)

- Simulator dodeljuje automatske ili implicitne korpe coverpoint varijablama
- Svaka različita vrednost variable dobija svoju korpu
 - na primer logic [2:0] addr ~ formira 8 korpi, za svaku vrednost po jednu
- Sadržaj dodeljene korpe biva inkrementiran kada se coverpoint osveži na definisani događaj.
- Imena korpama se dodeljuju automatski

```

module test;
  bit clk;
  logic [2:0] addr, data;
  ...
  ...
  covergroup cg1@(posedge clk);
    c1: coverpoint addr;
    c2: coverpoint data;
  endgroup
  ...
  ...
  cg1 cg_inst = new ;
endmodule

```

Automatske korpe primer

Name	Coverage	Goal	% of Goal	Status	Merge_inst	Gi
/test						
TYPE cg1	0.0%	100	0.0%	<div style="width: 100px;"></div>	1	
CVP cg1::c1	0.0%	100	0.0%	<div style="width: 100px;"></div>		
B bin auto['b000]	0	1	0.0%	<div style="width: 100px;"></div>		
B bin auto['b001]	0	1	0.0%	<div style="width: 100px;"></div>		
B bin auto['b010]	0	1	0.0%	<div style="width: 100px;"></div>		
B bin auto['b011]	0	1	0.0%	<div style="width: 100px;"></div>		
B bin auto['b100]	0	1	0.0%	<div style="width: 100px;"></div>		
B bin auto['b101]	0	1	0.0%	<div style="width: 100px;"></div>		
B bin auto['b110]	0	1	0.0%	<div style="width: 100px;"></div>		
B bin auto['b111]	0	1	0.0%	<div style="width: 100px;"></div>		
CVP cg1::c2	0.0%	100	0.0%	<div style="width: 100px;"></div>		

Eksplicitne korpe

- Korsnik može da formira eksplicitne korpe za svoje coverpoint varijable
- Time će se precizno pratiti samo korisnički definisane vrednosti tih varijabli
- Ključna reč bins se koristi za definisanje korpi
- Ovako definisane korpe se popunjavaju na definisani događaj ukoliko semplovane vrednosti jesu iz liste uz njih

Eksplicitne korpe primer

Name	Coverage	Goal	% of Goal	Status	Merge
/test					
TYPE cg1	0.0%	100	0.0%		1
CVP cg1::c1	0.0%	100	0.0%		
B bin a	0	1	0.0%		
B bin b	0	1	0.0%		
CVP cg1::c2	0.0%	100	0.0%		
B bin auto['b000]	0	1	0.0%		
B bin auto['b001]	0	1	0.0%		
B bin auto['b010]	0	1	0.0%		
B bin auto['b011]	0	1	0.0%		
B bin auto['b100]	0	1	0.0%		
B bin auto['b101]	0	1	0.0%		
B bin auto['b110]	0	1	0.0%		
B bin auto['b111]	0	1	0.0%		

```
module test;
bit clk;
logic [2:0] addr, data;

covergroup cg1@(posedge clk);
  c1: coverpoint addr{
    bins a = {1,2};
    bins b = {3,4,5,6,7};}
  c2: coverpoint data;
endgroup

cg1 cg_inst = new ;
endmodule
```

covergroup primer

- Coverpoint c1 je definisan za procenat pokrivenosti pojavljivanja opkodova
- Coverpoint c2 definisan za procenat pojavljivanja varijable addr
- Coverpoint c1 ima 3 eksplisitne korpe za proveru po grupama
- Coverpoint c2 dobija automatski 32 korpe ($2^5=32$)
- cg1test daje među pokrivenost između c1 i c2 coverpointa, koji sadrži $3 \times 32 = 96$ korpi
- Covergroup cg1 sadrži dakle coverpointe c1 i c2 i međuproizvod c1xc2
- cg1_inst predstavlja instancu kreiranu za covergroup cg1

```
enum logic [2:0] { HLT, SKZ, ADD, AND, XOR, LDA, STO, JMP} opcode;
logic [4:0] addr;
covergroup cg1 @(posedge clk);
    c1: coverpoint opcode {
        bins low  = {HLT, SKZ };
        bins mid = {ADD, AND, XOR };
        bins high = {LDA, STO, JMP};
    } cg1test : cross c1, c2 ;
    c2: coverpoint addr ;
endgroup
cg1 cg1_inst = new();
```

Name	Coverage	Goal	% of Goal	Status	Merge
/alu_test					
TYPE cg1	37.1%	100	37.1%	██████████	1
CVP cg1::c1	100.0%	100	100.0%	██████████	
B) bin low	3	1	300.0%	██████████	
B) bin mid	6	1	600.0%	██████████	
B) bin hig	5	1	500.0%	██████████	
CVP cg1::c2	7.8%	100	7.8%	██████████	
CROSS cg1::cg1test	3.6%	100	3.6%	██████████	

eksplicitne korpe

```
module test;
  bit clk; logic [3:0] addr, data; int i =1;

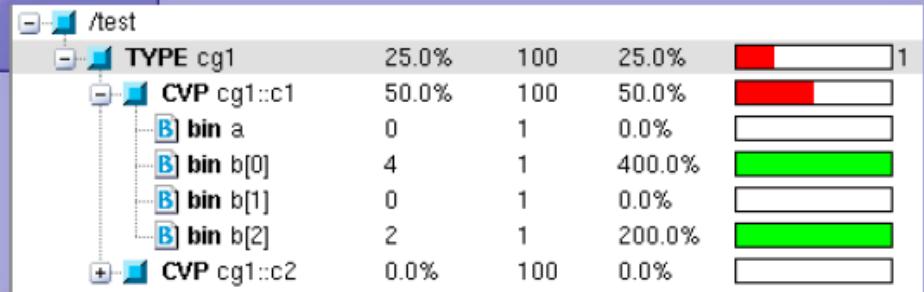
  covergroup cg1@(posedge clk);
    c1: coverpoint addr{
      bins a = {1,2};
      bins b[3] = { 7,8,9,10, 3,4,5,6, 11,12,13,14};}
      //bins b[] = { 7,8,9,10, 3,4,5,6, 11,12,13,14};}
    c2: coverpoint data; // automatic bins by simulator
  endgroup
  cg1 cg_inst = new ;

  initial for (int k=0;k<20;k++) begin #2ns clk = ~clk; end

  always @(posedge clk) i <= i + 1;
  always @(posedge clk) if ( i == 4 ) begin addr <= 7; end
  always @(posedge clk) if ( i == 6 ) begin addr <= 9; end
  always @(posedge clk) if ( i == 8 ) begin addr <= 12; end
  always @(posedge clk) if ( i == 10 ) begin addr <= 11; end

endmodule
```

- “bins a” predstavlja jednu korpu koja pokriva dve vrednosti
- “bins b[3]” predstavlja niz od 3 korpe.
 - Izlistane vrednosti u zagradama biće uniformno raspodeljene na te 3 korpe. Ukoliko broj vrednosti nije deljiv sa 3, poslednja korpa preuzima ostatak.
- “bins b[]” bez indeksa, se svodi na jednu korpu po svakoj vrednosti iz liste



Korpe za praćenje tranzicija

- Broji se broj tranzicija iz jednog stanja u drugo
- Operator “=>” definiše sekvencu tranzicija
- $x =>z =>z$ označava x kog prati y kog prati z

```
logic [3:0] addr;

covergroup cg1@(posedge clk);
    c1: coverpoint addr{
        bins b1 = (4=>5=>6),([4:6],7=>8,9); // 4=>5=>6 or (any of 4 5 6), or 7=>8 or 9
        bins b2 = ([7:9],10=>8,12);           // any value of 7 8 9 or 10=>8 or 12
        bins b3 = (3[* 5]);                   // 3=>3=>3=>3=>3, repetition
        bins b4 = (3[* 3:5]);                // 3=>3=>3  3=>3=>3=>3  3=>3=>3=>3=>3
        bins b5 = (3[->3]);                 // ...3=>...=>3...=>3  ( ... any values in between )
        bins b6 = (3[=2]);                  // ...3=>...=>3...=>3  ( ... any values in between )
        wildcard bins b7 = {4'b11??};       // ?? acts as wildcard for remaining bits
    }
endgroup
cg1 cg_inst = new ;
```

Covergroup u klasama

- Moguće je definisati covergroupe kao članice klase
- Time se može definisati model pokrivanja za protected i lokalne članice klase
- U primeru prvo kreiramo klasu obj
- Zatim kreiramo covergroup koji prati članicu klase obj.m nalik senistiviti listi
- Pri tome se prati pokrivenost članice ma

```
module test;
    class class1;
        logic [2:0] m;
    endclass

    class myclass ;
        class1 obj;
        logic [1:0] ma;
        covergroup cg1 @(obj.m);
            coverpoint ma;
        endgroup

        function new(); // order is important
            obj = new ; // First create class instance
            cg1 = new ; // Second coverage instance
        endfunction
    endclass

    myclass c1 = new ;
endmodule
```

Međupokrivenost

- Praćenje među proizvoda
- Coverpoints unutar covergroupa
- Automatski se kreiraju korpe za međupokrivenost
- Ključna reč cross definiše međupokrivenost

```
logic [1:0] addr, data;  
covergroup cg1@(posedge clk);  
    c1: coverpoint addr;  
    c2: coverpoint data;  
    c1xc2 : cross addr, data;  
endgroup  
cg1 cg_inst = new ;
```

	C1.auto[0]	C1.auto[1]	C1.auto[2]	C1.auto[3]
C2.auto[0]				
C2.auto[1]				
C2.auto[2]				
C2.auto[3]				

Međupokrivenost - primer

```
bit clk; logic [1:0] addr, data; int i =1;

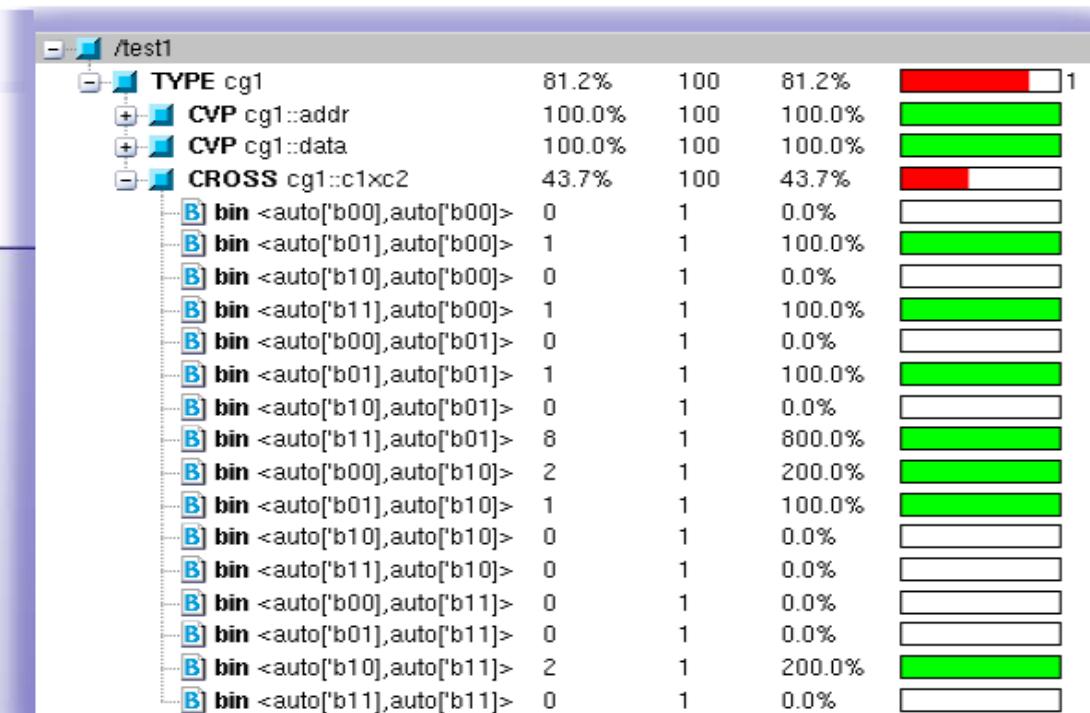
covergroup cg1@(posedge clk);
    c1xc2 : cross addr, data;
endgroup

cg1 cg_inst = new ;

initial for (int k=0;k<40;k++) #2ns clk = ~clk;

always @(posedge clk) i <= i + 1;

always @(posedge clk) if ( i == 4 ) begin addr <= 1; data <= 1; end
always @(posedge clk) if ( i == 5 ) begin addr <= 2; data <= 3; end
always @(posedge clk) if ( i == 6 ) begin addr <= 3; data <= 0; end
always @(posedge clk) if ( i == 7 ) begin addr <= 0; data <= 2; end
always @(posedge clk) if ( i == 8 ) begin addr <= 2; data <= 3; end
always @(posedge clk) if ( i == 9 ) begin addr <= 3; data <= 1; end
always @(posedge clk) if ( i == 10) begin addr <= 0; data <= 2; end
always @(posedge clk) if ( i == 11) begin addr <= 1; data <= 0; end
always @(posedge clk) if ( i == 12) begin addr <= 1; data <= 2; end
always @(posedge clk) if ( i == 13) begin addr <= 3; data <= 1; end
```



Covergroup opcije

- Opcije opredeljuju ponašanje covergoupa, coverpointa i međuproizvoda
- Postoje dve vrste opcija za covergoupe
 - Instance specific (specifične za instancu covergroupe)
 - Type specific (specificira opcije covergroup u celini)

Field type	Option name	Default	cg	cp	cc	Definition
int	weight	1	Y	Y	Y	Specifies weight of this element of cover group for calculating overall cumulative coverage
int	goal	100	Y	Y	Y	Target goal for covergroup, coverpoint or cross
string	comment	" "	Y	Y	Y	Comment to include in coverage report
bit	strobe	0	Y	--	--	If set 1, all samples happen at the end of time slot (deferring the sample)

Covergroup opcije

Field type	Option name	Default	cg	cp	cc	Definition
string	name	unique	Y	--	--	Instance Name
int	weight	1	Y	Y	Y	Specifies weight of this element of cover group for calculating overall cumulative coverage
int	goal	100	Y	Y	Y	Target goal for covergroup, coverpoint or cross
string	comment	" "	Y	Y	Y	Comment to include in coverage report
int	at_least	1	Y	Y	Y	Minimum number of hits for each bin
bit	detect_overlap	0	Y	Y	--	If true, issue warning for values overlapping bins
int	auto_bin_max	64	Y	Y	--	Max number of auto bins for coverpoint
bit	per_instance	0	Y	--	--	If true, track coverage for each instance
int	cross_num_print_missing	0	Y	--	Y	Max number of uncovered cross bins to report
int	cross_auto_bin_max	No bound	Y	Y	Y	Max number of auto bins for cross

Covergroup opcije - primer

```
module cp_inst_options ;
    bit clk;
    logic [1:0] addr; int i = 1;
    logic [3:0] data;

    covergroup cg1@(posedge clk);
        option.per_instance = 1 ;
        cp1 : coverpoint addr {
            bins A[] = {[0:3]};
            option.comment = "This Option will appear on the Cov Report";      }
        cp2 : coverpoint data {
            bins low = {[0:5]};
            bins mid = {[6:15]};
            option.comment = "This Option message in cp2 will appear on the Cov Report";}
        cp3 : coverpoint data {
            // create 128 automatic bins for coverpoint cp3
            option.auto_bin_max = 128;
            option.weight = 2;      }
        addrxdata : cross cp1,cp2 {
            bins x1 = binsof(cp2) intersect {[8:10]};
            bins x2 = binsof(cp1) && !binsof(cp2.mid);
        }
    endgroup : cg1
endmodule
```

Covergroup metode

return type	Methods	cg	cp	cc	Description
void	sample()	Y	--	--	Triggers sampling of covergroup (forces)
real	get_coverage()	Y	Y	Y	Calculate the coverage for type (0..100)
read	get_inst_coverage()	Y	Y	Y	Calculate the instance coverage (0..100)
void	set_inst_name(string)	Y	--	--	Sets the instance name
void	start()	Y	Y	Y	Starts collecting coverage information
void	stop()	Y	Y	Y	Stops collecting coverage information
real	query()	Y	Y	Y	Returns the coverage information for the covergroup
real	inst_query()	Y	Y	Y	Returns the coverage information for this specific instance

Pokrivenost bitni detalji

- UCDB (United Coverage Database) format je definisao Mentor i predao Acceleri/IEEE
 - Alat prikuplja sve podatke i nad njima pravi analizu
- Podrška pokrivanju od strane alata
 - Podrška prikupljanju svih tipova coverage-a
 - GUI izveštavanje
 - Omogućavanje/onemogućavanje bilo kog tipa coverage-a
- Ograničenja code coverage-a
 - Coverage ne zna ništa o našem dizajnu
 - Ne može se otkriti šta nedostaje u kodu
 - Samo nam govori o kvalitetu implementacije
 - Čak i ako smo aktivirali sve linije u kodu, ne znači da smo ih sve testirali
 - 100% code coverage je potreban, ali ne i dovoljan uslov testiranja dizajna

Statička verifikacija

- Simulacija je dobra koliko i stimulus u njoj kreiran (lanac je jak koliko najslabija karika)
 - Nedovoljan stimulus → neaktivirani bagovi
 - Aktivirani bagovi moraju da ispropagiraju sve do tačke uočavanja
- Statičkim alatima potreban je RTL kod ili netlista
 - Bez testbenča → ranije se može pokrenuti
 - Provera koda po određenim pravilima

Tipovi statickih alata

1) RTL Lint

- Formirana pravila prikupljana godinama iskustva kako pisati dobar RTL kod
- Proverava se
 - Naming conventions (imenovanje signala modula...)
 - Lečevi
 - Nekompletne liste osetljivosti
 - Nesintetizibilne konstrukcije
 - Izostavljeni reset...

2) Statička provera

- Pseudo sinteza za proveru strukture dizajna
- proverava:
- CDC – klok domen krosing
- Napomske nivoe (level shifters)
- Kombinacione petlje
- Multiple drivers
- Mrtav kod, nikad aktivirana FSM stanja

3) Sekvencijalna formalna provera

- Provera netlist reprezentacije dizajna iz pseudo sinteze ili prave sintetisane netliste
- Ograničenja okruženja (SDC file)
Synopsys Design Constraint je postao svetski prihvaćen standard za ograničavanje dizajna
- sekvenca inicializacije
- Između ivica takta proveravaju se stanja dizajna
- Fokus je na funkcionalnosti, ne na konekcijama

- Propagacija neinicijalizovanih stanja
- FSM-ovi sa mrtvom petljom ili izolovanim stanjima
- Zaglavljivanje zbog ograničenja
- Redundantan ili mrtav kod

4) Automatizovane formalne provere

- Assertions koje se proveravaju su generisani automatski
- Neke primene:
- LEC: Logic Equivalence Checking
 - RTL naspram netliste
 - Netliste naspram netliste
- Static Timing Analysis (STA) Statička vremenska analiza
 - Pronalazi multi cycle paths, false paths i clock domain crossings. To su putanje sa više ciklusa takta, lažne putanje (kod kojih takt nije od značaja), putanje između različitih izvora takta
 - STA se mora konfigurisati po pitanju ovih pretraga.

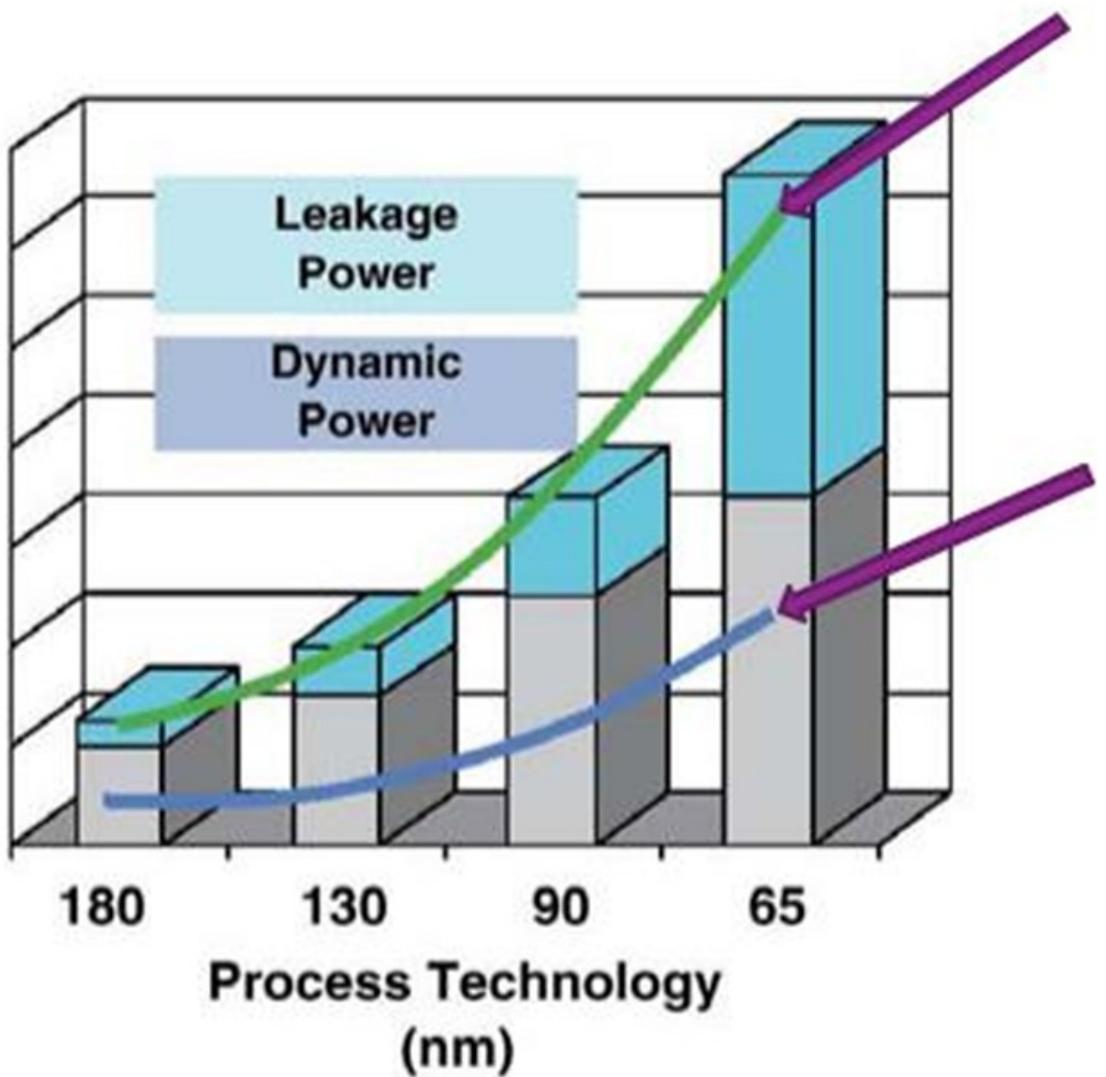
5) Verifikacija formalnih karakteristika

- Proverava da li su assertions always/never true
- Koristi formalne karakteristike pri verifikaciji kritičnih tačaka:
 - Resurse u kontrolnoj logici
 - Interfejsima dizajna
 - Konačnim automatima

Power Island simulacije

- Kritičan segment simulacione verifikacije svakako predstavlja menadžment energije u integrisanom kolu.
- Ukoliko svi delovi kola stalno rade (makar u praznom hodu) potrošnja energije je velika
- Ideja je da se tokom relativno dužeg ne korišćenja određenih blokova oni isključuju.

Problem potrošnje energije



- U savremenim Cmos tehnologijama sve je veće učešće statičke potrošnje(65nm i niže)
- Tu dominiraju struje curenja koje su sve veće što su manje dimenzije tranzistora

Statička i dinamička potrošnja

- Problem statičke disipacije rešava se i snižavanjem napona napajanja, koji sticajem okolnosti opada sa minimizacijom Cmos dimenzija.
 - (45nm : 1V), (16nm : 0.5V-0.8V), (7nm : 0.4V),
 - Izuzetno visoke struje daju veliku potrošnju bez obzira na sve niže nivoje napajanja
 - V_{thmos} suštinski zavisi od dimenzija FET-a, manje dimenzije – niži napon

- Unified Power Format (UPF) predstavlja IEEE standardizovan pristup za optimizaciju potrošnje iz perspektive digitalnog dizajna
- Tema prevazilazi okvire ovog kursa, ovde je navedene radi opštosti

Assertion (na tvrdnji) bazirana verifikacija

- Tredovi i među procesna komunikacija
 - Događaji, Semafori, Mailboxes
- Uvod u Assertions
 - SV-Assertions
 - SVA biblioteke
 - Assertions, Property checkers, Sekvence
 - Osnovne i napredne SVA konstrukcije
 - Assertion coverage.
- Direktni Programerski Interface
- Formalna Verifikacija

Događaji – events

- Identifikator deklarisan kao event tip podatka još se naziva *named event (imenovani događaji)*
 - event e1;
- U Verilogu, trigerovanje događaja se inicira sa “->” i “@”
- Verilog događaji su statički; SV događaji su dinamički
- Imenovani događaji i kontrola događaja
 - Način opisivanja komunikacije/sinhronizacije između dva ili više konkurentno aktivnih procesa
- Događaji mogu biti blokirajući ili ne blokirajući
- SV događaji se ponašaju kao rukovaoci sinhronizacionim redovima izvršenja, prosleđeni kao argumenti taskovima

Okidanje događaja

- Imenovani događaji se okidaju korišćenjem operatora “->”
- Okidanjem događaja deblokiraju se svi procesi koji su u stanju čekanja na njega.
- Kada se desi okidanje
 - Stanje okidanja nije opservabilno, već samo njegovi efekti
- Slično kao ivično okidani FF
- konstrukcija if (posedge clk) // pogrešno okidanje
- konstrukcija @(posedge clk) // ispravno okidanje

Primer okidanja događaja

```
module main;
  event e;
  initial begin
    #20 ;
    -> e ; // Blocking event trigger, immediate assignment
    $display (" e is triggered at %t ",$time);
  end
  initial #40 $finish;
  always @e // This process should be running before even event e is triggered !!
    $display (" event triggered at ", $time); endmodule
```

e is triggered at 20
event triggered at 20

Ne blokirajuće okidanje

- Ne blokirajuće okidanje pomoću “->>” operatora
- “->” Blokirajuće : trenutna dodela
 - >> Ne blokirajuća : dodata unutar NBA regiona
(sećamo se schedulinga u SV)
- Izvršava se bez blokiranja
- Događaj će biti okinut unutar NBA regiona
- Koristi se da prevenira race condition između procesa

Okidanje primer

```
module test1 ();
    event e; bit temp, clk = 1'b0 ;

    initial for (int i=0; i <6; i++) #1ns clk = ~clk;

    always @e
        $display (" always block ", temp, $time);

    initial begin
        temp <= 1; // NBA
        -> e; // Blocking event
        $display (" from initial block", temp, $time);
    end
endmodule
```

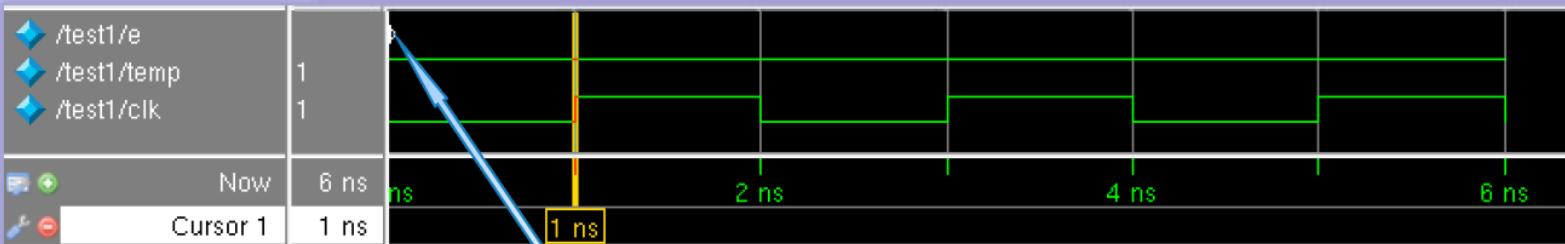
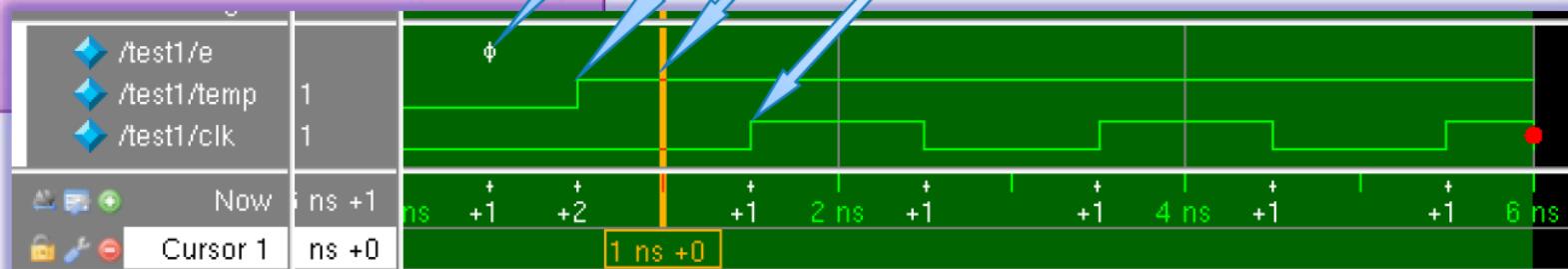
from initial block 0 0
always block 0 0

Blocking assignment of e

Temp gets assigned in NBA

1ns time

Posedge of clk, @ 1ns + 1



event happens at 0 time

Okidanje primer 2

```
module test2 ();
    event e;
    bit temp, clk = 1'b0;

    initial for (int i=0; i <6; i++) #1ns clk = ~clk;

    always @e
        $display (" always block ", temp, $time);

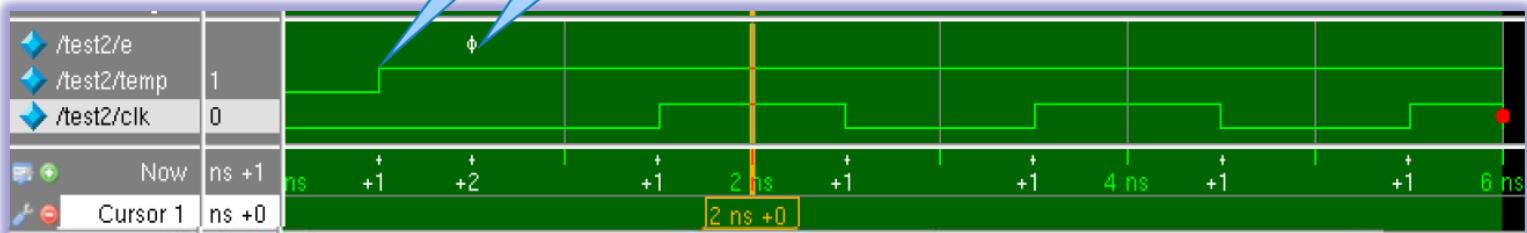
    initial begin
        temp <= 1; // NBA
        ->> e;      // NBA event
        $display (" from initial block", temp, $time);
    end
endmodule
```

from initial block 0 0
always block 0 0

Ordering of the threads matter here!!

Temp gets assigned in NBA

Non Blocking assignment of e



Okidanje primer 3

```
module test3 ();
    event e;
    bit temp, clk = 1'b0;

    initial for (int i=0; i < 6; i++) #1ns clk = ~clk;

    always @e
        $display (" always block ", temp, $time);

    initial begin
        temp = 1; // BA for temp
        ->> e; // NBA event
        $display (" from initial block", temp, $time);
        ->> @(posedge clk) e;
        // -> @(posedge clk) e; // ERROR, Not allowed !
    end
endmodule
```

from initial block 1 0
always block 1 0
always block 1 1

Blocking assignment of **e** at a event
NOT possible !!!

Non Blocking assignment of **e**

Non Blocking assignment
of **e** @**posedge** of **clk**



@, operator čekanja na događaj

- @, operator čekanja na događaj
 - @ operator blokira pozivajući proces sve dok se događaj na koji se čeka ne okine
 - Okidanje će deblokirati proces, **ako i samo ako** je proces počeo da čeka pre trenutka okidanja
 - Ako se okidanje desi pre nego što proces na njega počne da čeka, tada će čekajući proces ostati blokiran
- wait (event_identifier.triggered)
 - Korišćenjem ovakvog čekanja, okidanje će deblokirati čekajući proces, bez obzira da li je čekanje počelo pre ili istovremeno kad i okidanje.
 - proces koji čeka sa @ operatorom može da propusti okidanje ukoliko se desi race condition
 - Može ostati da čeka sve dok se ne desi naredno okidanje
- Wait operator nikada neće propustiti okidanje, ukoliko je čekanje započelo ranije ili istovremeno sa okidanjem

@ nasuprot wait(event.triggered)

```
module trig;  
  event e1,e2;  
  initial begin  
    fork  
      -> e1 ;  
      @e1 $display(" e1 ");  
      -> e2;  
      wait(e2.triggered)  
        $display(" e2 ");  
    join  
    $display (" all done ");  
  end  
endmodule
```

X

e2

```
module trig;  
  event e1,e2;  
  initial begin  
    fork  
      ->> e1 ;  
      @e1 $display(" e1 ");  
      -> e2;  
      wait(e2.triggered)  
        $display(" e2 ");  
    join  
    $display (" all done ");  
  end  
endmodule
```

Y

e2
e1
all done

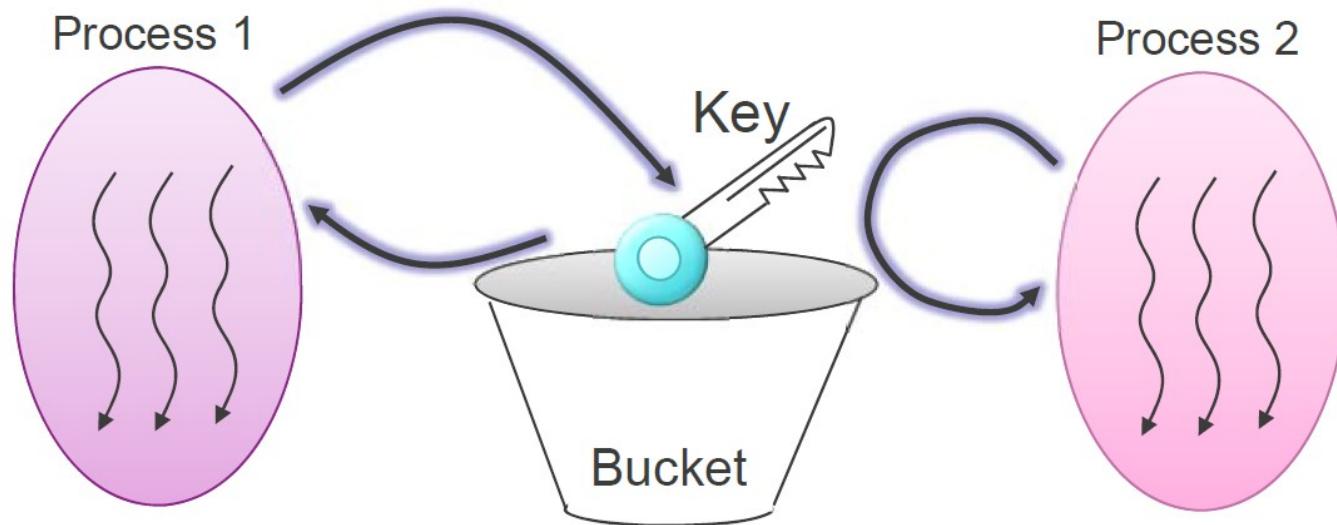
```
module trig;  
  event e1,e2;  
  initial begin  
    fork  
      @e1 $display(" e1 ");  
      -> e1 ;  
      -> e2;  
      wait(e2.triggered)  
        $display(" e2 ");  
    join  
    $display (" all done ");  
  end  
endmodule
```

Z

e2
e1
all done

Semafori

- Semafor se može posmatrati kao tehnika razrešavanja pristupa resursu koji je deficitaran (više korisnika želi da pristupi resursu istovremeno) koncept tokena/ključa
 - Jednostavno rešenje, praktično preuzeto iz koncepta operativnog sistema
- Proces preko semafora dolazi do ključa pre nego što nastavi da se izvršava
- Kada se proces izvrši, preko semafora vraća ključ
- Ukoliko je broj ključeva mali, ostali procesi čekaju na povratak prvog ključa!
- Obezbeđuje kontrolu pristupa deljenim resursima, sinhronizaciju
- Kreira se i rasformira dinamički



- Semafori su klasa Sistem Veriloga
- Mogu se koristiti u postojećem osnovnom obliku ili se iz njih izvoditi nove klase
 - Pri kreiranju semafora definiše se broj ključeva new (int keyCount)
 - Uzimanje ključa –jednog ili više: get (int keyCount)
 - Vraćanje ključa –jednog ili više : put (int keyCount)
 - Pokušaj uzimanja jednog ili više ključeva bez blokiranja: try_get(int keyCount)

```
class semaphore;  
    local chandle p1;  
    local int keyCount;  
    function new(int keyCount = 0);  
        this.keyCount = keyCount;  
    endfunction  
    extern function void put( int keyCount = 1);  
    extern task get(int keyCount = 1);  
    extern function int try_get( int keyCount = 1);  
endclass
```

`new():`

- **function** `new (int keycount = 0); //default count 0`
- `keycount` specifies number of keys initially allocated

`put():`

- **task** `put (int keycount = 1); //default count 1`
- `put()` returns specified number of keys to semaphore
- waiting process can grab these keys

`get ():`

- **task** `get (int keycount = 1); //default count 1`
- if specified number of keys are not available, process blocks !

`try_get():`

- **function int** `try_get (int keycount =1); //default count 1`
- if specified number of keys available, method returns 1 and execution continues
- if not, method returns 0.

Semafori primer

```
module sema;
    int count;
    semaphore sem = new(1);

    initial begin : P1
        sem.get;
        $display ("in P1",++count);
        sem.put; // what happens if commented !
    end

    initial begin : P2
        #1; // what happens if commented !
        sem.get;
        $display ("in P2",++count);
        sem.put;
    end
endmodule
```

in P1 1
in P2 2

- sem = new (1)
 - Novi semafor po imenu sem je kreiran sa jednim ključem
- P1 i P2 procesi startuju istovremeno
- P1 uzima ključ preko semafora "sem"
- P2 će biti blokiran ako pokuša da uzme ključ u isto vreme
- \$display i ++count je izvršen unutar procesa P1, vraćen je ključ "sem.put"
- Nakon #1, P2 nastavlja i traži ključ
- P2 uzima ključ (vraćen od strane P1)
- Izvršava svoj \$display i ++count, vraća ključ.
- Šta biva ako se crvene linije zakomentarišu, prvo jedna pa druga

Mailbox (sanduče)

- mailbox: komunikacioni mehanizam koji omogućava razmenu poruka između procesa
- Slično pravim poštanskim sandučićima
 - Pismo se isporuči u sanduče, a primalac ga preuzima kasnije
 - Ako je sanduče prazno, primalac odlučuje da li će da trajno čeka, ili će proveravati povremeno!
 - Ako je sanduče puno, poštar (zapravo pošiljalac) čeka dok se sanduče ne isprazni
- mailbox je klasa u SV
 - Kreiranje mailbox-a: new()
 - Postavljanje poruke u sanduče: put()
 - Pokušaj ostavljanja poruke u sanduče bez blokiranja : try_put()
 - Preuzmi poruku iz sandučeta: get()
 - Kopiraj poruku iz sandučeta, bez uzimanja (i dalje ona ostaje u sandučetu): peek()
 - Pokušaj preuzimanja/kopiranja poruke iz sandučeta bez blokiranja: try_get() / try_peek()
 - Proveri broj poruka u sandučetu: num()

```
new() :function new (int bound = 0); // default bound 0
```

mail box constructor, optionally specifies the max size

```
num() :function int num ();
```

- returns the number of messages currently in mailbox

```
put() :task put (singular message);
```

- places a message in mailbox; blocks if mailbox full

```
try_put() :function int try_put (singular message);
```

- places a message in mailbox; returns 0 if mailbox full

`get() : task get (ref singular message);`

- retrieves message from mailbox; blocks if mailbox empty; error if type mismatch

`try_get() : function int try_get (ref singular message);`

- retrieves message from mailbox; returns 0/+1/-1; empty/success/type mismatch

`peek() : task peek(ref singular message);`

- copies message from mailbox; blocks if mailbox empty; error if type mismatch

`try_peek() : function int try_peek (ref singular message);`

- copies message from mailbox; returns 0/+1/-1; empty/success/type mismatch

Parametrizovani mailbox

- Podrazumevano, mailbox nema definisani tip (misli se na tip poruka koje će se prebacivati)
- Može doći do problema ako se tipovi poruka razlikuju
- pomoću parametrizacije, definiše se specifičan tip poruka
 - `mailbox #(string) str_mailbox = new (2);`
- Kompajler detektuje nesaglasne tipove u ovom slučaju u toku kompajliranje, dok u slučaju sa generičkim mailboxovima to se otkriva tek u toku izvršavanja programa

```
typedef enum {s1,s2,s3,s4} state_s ;  
mailbox #(state_s) enum_mailbox = new ;
```

```
mailbox #(string) str_mailbox = new ;
```

```
mailbox #(int) str_mailbox = new ;
```

Mailbox primer

```
program mail ;  
mailbox my_mailbox = new(4);  
initial begin  
if (my_mailbox) begin  
    fork  
        put_data();  
        get_data();  
    join  
end  
end  
endprogram
```

```
task put_data();  
integer i;  
begin  
    for (i=0; i<6; i++) begin  
        #2;  
        my_mailbox.put(i);  
        my_mailbox.put("xyz");  
        $display("Writing data %d @time %d",i, $time);  
    end  
end  
endtask
```

- 2 writes into mailbox, 1 read from mail box
- Mailbox message any data-type

```
task get_data();  
string rdata;  
begin  
    for (int i=0; i<6; i++) begin  
        #2;  
        my_mailbox.get (rdata);  
        $display ("Reading Data %s @time %d", rdata, $time);  
    end end  
endtask
```

Writing data	0	@time	2
Reading Data	0	@time	2
Writing data	1	@time	4
Reading Data	xyz	@time	4
Writing data	2	@time	6
Reading Data	1	@time	6
Reading Data	xyz	@time	8
Writing data	3	@time	8
Reading Data	2	@time	10
Reading Data	xyz	@time	12
Writing data	4	@time	12

pregled

- Događaji (Event)
 - Blokirajuće okidanje
 - Ne blokirajuće okidanje
- @ naspram event.triggered
- Semafori
 - Semaphore metode
- Mailbox
 - Mailbox metode
 - Parametrzovani Mailboxes