

Sistem Verilog

Scheduling, komunikacija, OOP

SV raspoređivač (scheduling)

- Osnovni problem leži u upotrebi sekvencijalnog računarskog sistema koji simulira HW paralelne prirode.
- Važno je razraditi mehanizme za emulaciju / verifikaciju kombinacionih kola
- Važno je razraditi mehanizme za emulaciju / verifikaciju sekvencijalnih kola

Race condition (trka)

- Race condition se vrlo lako kreira, razume, dokumentuje, ali se teško otkriva
- Dva izraza raspoređena da se izvrše istovremeno, ako redosled nije definisan proizvode Race condition
- Nekad se može detektovati na vremenskom dijagramu simulatora kao glic (ni 0 ni 1)
- Prevencija Race Condition-a !
 - Postoje detalji koji i dalje nisu specificirani u simulatorima (Možda manje od 1%)
 - Korišćenje različitih simulatora možda može da pomogne
 - Treba se držati osnovnih prostih pravila za digitalni dizajn (design guidelines)
 - Treba pažljivo testirati dizajn
 - HDL jezici su po prirodi paralelni
 - Ali, Procesor koji simulira je sekvencijalan
 - A Multi procesorski sistemi, multi core? (Sinhronizacija vrlo problematična)

Write race

- Multiple driver (višestruki upis) !
- Situacija nije determinisana standardom, IEEE Std 1800-2017.
- Rešenje ostavljeno simulatoru.

```
module race(clk) ;  
    input clk;  
    reg [2:0] a;  
    always @(posedge clk)  
        a = 5;  
    always @(posedge clk)  
        a = 1;  
endmodule
```

a gets 'x'

Read race

```
function incr();
    g_count = g_count++;
endfunction
```

```
always @(clk)
    count1 = incr();
```

```
always @(clk)
    count2 = incr();
```

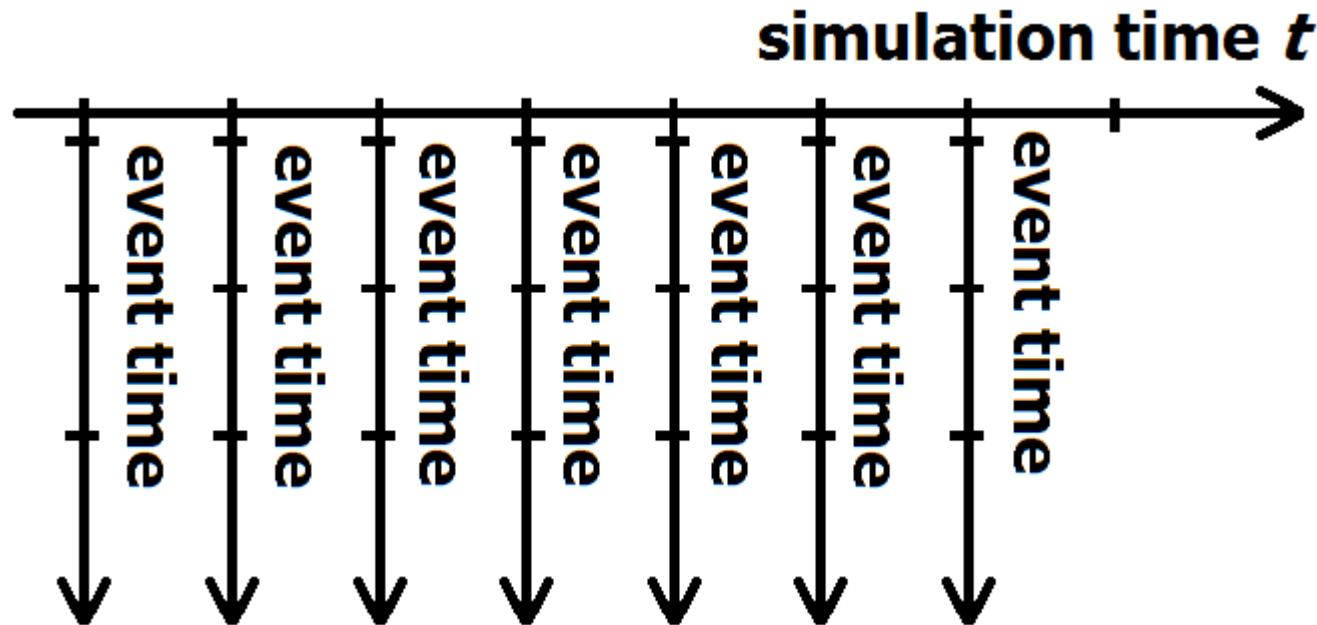
Uvodimo pojam događaj (event)

- Događaj promene vrednosti čvora, varijable, predstavlja događaj (event)
- Procesi su osetljivi na događaje promene vrednosti
- Kada se desi događaj promene vrednosti, svi procesi osetljivi na njega sračunavaju svoju vrednost u slučajnom redosledu pozivanja.
- Sračunavanje vrednosti u preocisu je tekoće događaj (event)
 - Vreme događaja
 - Simulaciono vreme

Koncept vremena u SV simulatoru

- Pošto je neophodno u SW modelovati paralelizam HW, nedovoljna je samo jedna dimenzija vremena “ t ” (simulaciono vreme – simulation time)
- Dodaje se još jedna dimenzija “vreme događaja” – event time
 - Simulator vodi računa o redosledu događaja i izvršava ih u skladu sa redosledom
 - Događaji se postavljaju u red izvršenja događaja (event queue)
 - Dodavanje događaja u red izvršenja naziva se raspoređivanje događaja (event scheduling)

- U svakom koraku simulacionog vremena razrešava se redosled “vremena događaja”



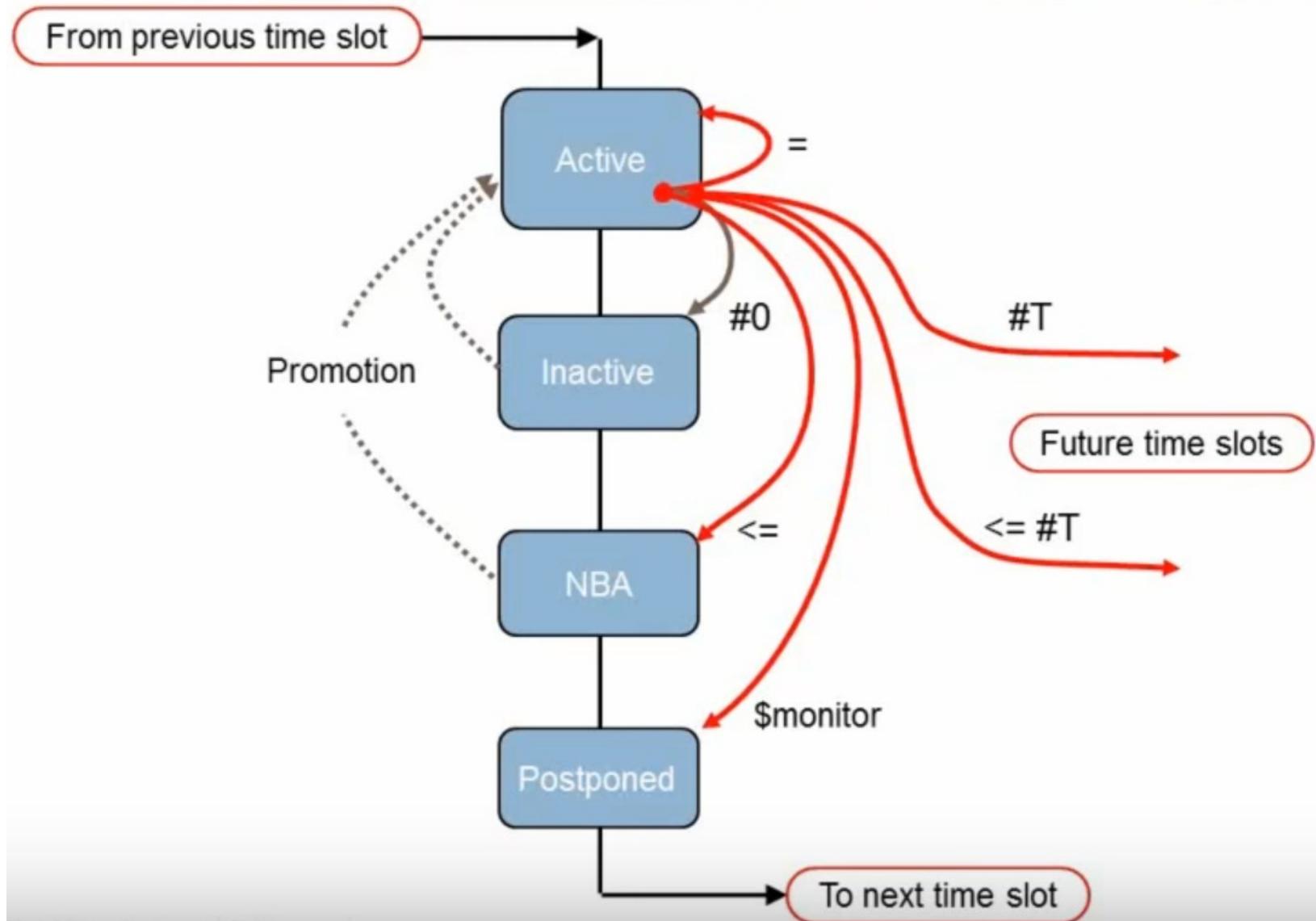
- Simulaciono vreme se menja u koracima definisane vremenske rezoluicije.
- u svakom koraku simulacionog vremena završavaju se sve simulacione aktivnosti vremena događaja koje teče iz te tačke.
- Kada se urade svi događaji vezani za tu tačku, simulaciono vreme može da se inkrementira

Raspoređivanje događaja

- Simulator obezbeđuje dinamičko raspoređivanje događaja
 - Raspodela u vremenu
- Svaki događaj ima samo jedno simulaciono vreme u kom se izvršava
- Raspoređivanje događaja u SV je unazad kompatibilno sa Verilogom.
- Semantika SV je definisana za Event-Driven simulacije
 - Razlikujemo blokirajuće (Blocking) dodele
 - i ne blokirajuće (Non-Blocking) dodele
- Blokirajuće: trenutna dodata u okviru niti izvršenja
`temp = 1;`
- Ne blokirajuća: izvršenje nakon što se niti izvršenje suspenduju
`temp <= 1;`

- Bitno je prevenirati race-condition između dizajna i test-benča
- Uključiti SVA (SV assertions)
- Obezbediti programske blokove, clocking blokove (blokove takta) i dizajn module da funkcionišu harmonično

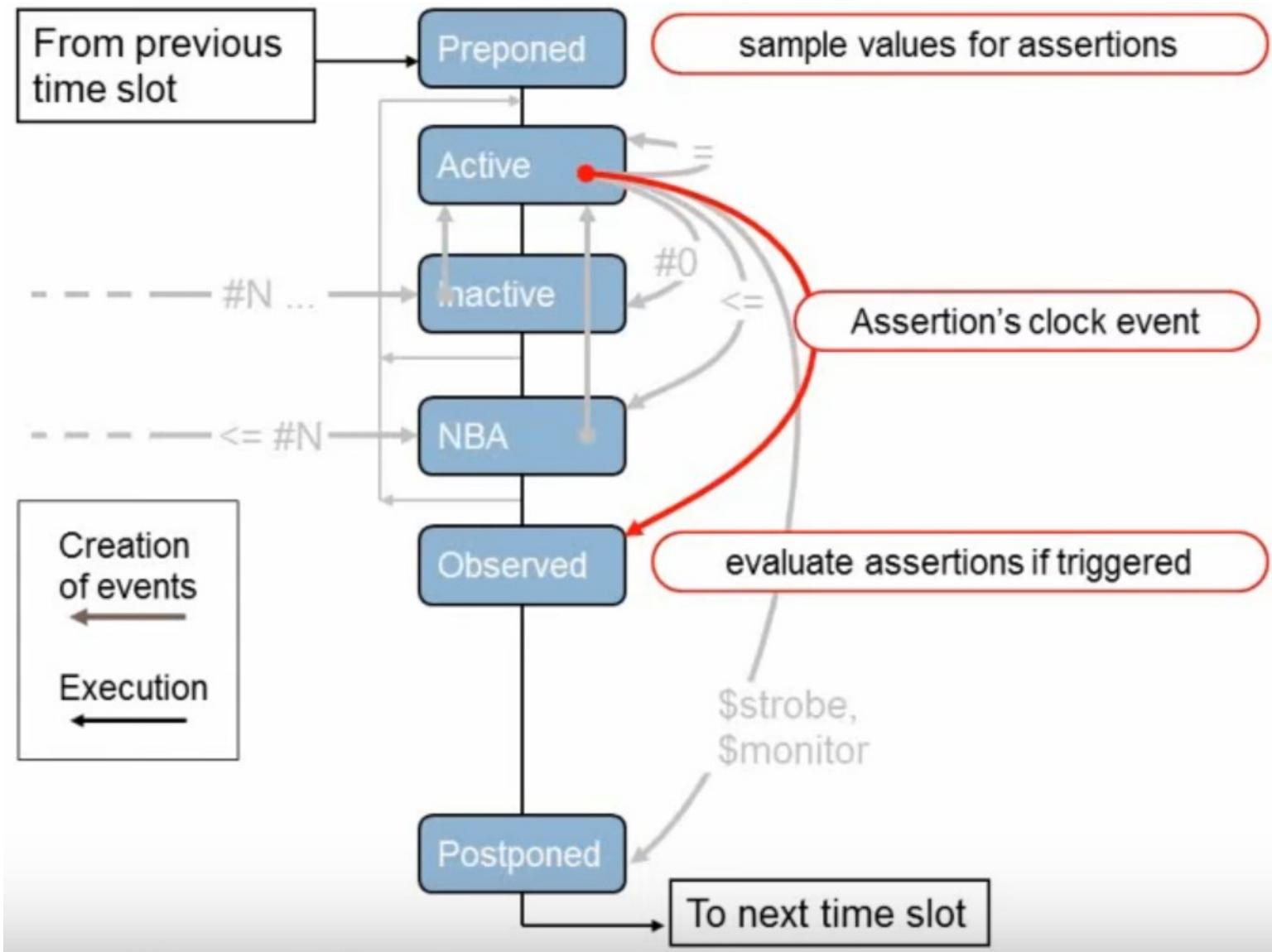
Verilog rasporedjivač



Verilog raspoređivač tumačenje

- Active
 - Realizuje blokirajuću dodelu = prioritetno
- Inactive
 - Realizuje zakasnelu blokirajuću dodelu #0
- NBA
 - Realizuje ne blokirajuću dodelu <=
- Postponed
 - \$monitor koji se koristi za praćenje toka simulacije i raportiranje

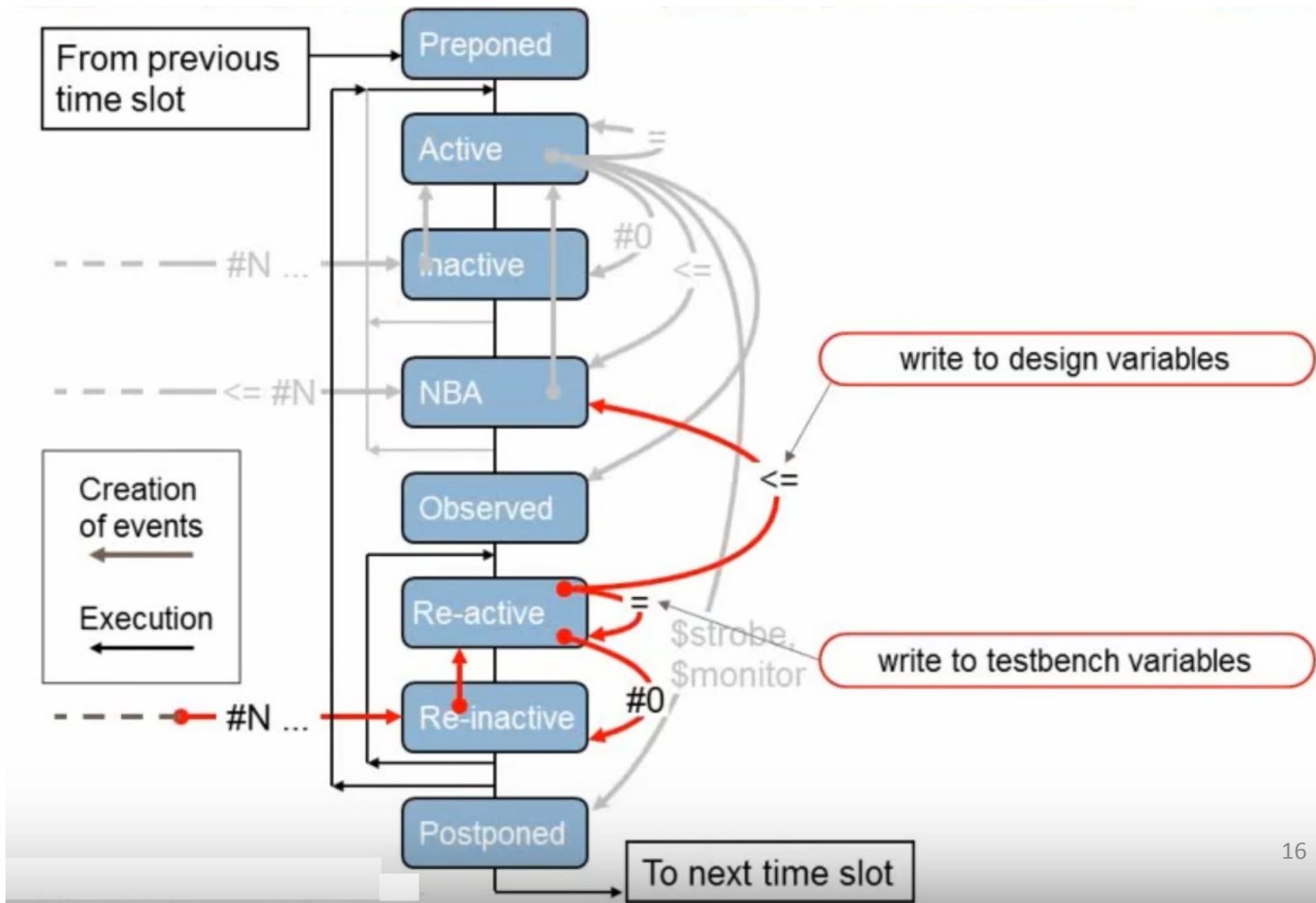
SV rasporedjivanje



SV raspoređivač tumačenje

- Novi blokovi nadograđeni na Verilog scheduler
 - Preponed
 - Ovaj blok dozvoljava PLI (programming language interface) pristup podacima u datom vremenskom koraku – time slotu
 - Tipično preuzima vrednost signala potrebnih za assertion analizu
 - Observed
 - Sračunavanje assertion-a ukoliko su trigerovani (na primer taktom ili događajem na drugom signalu)

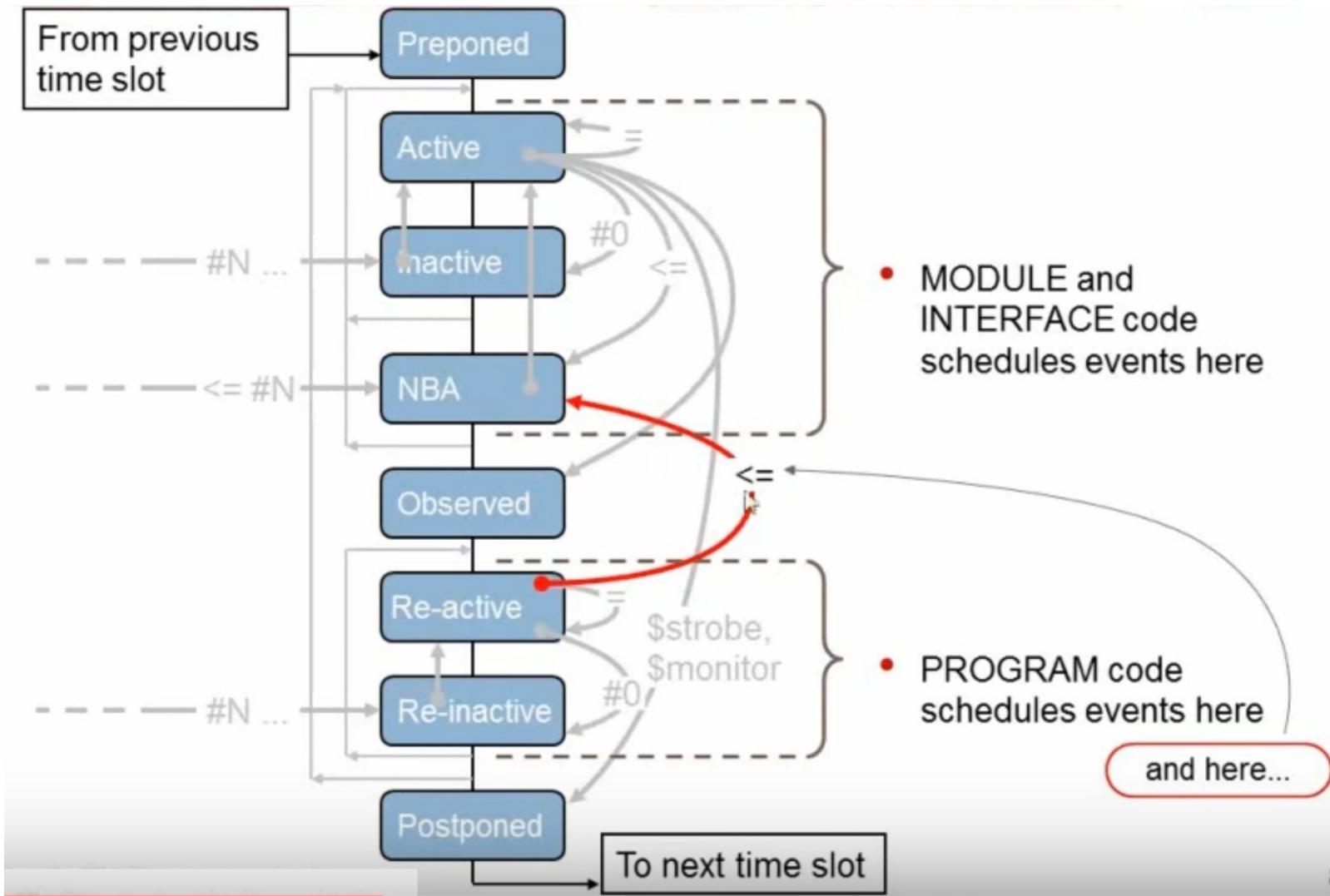
SV raspoređivanje za TB pokrivanje



Tumačenje proširenja

- Nakon Observed bloka dodati su blokovi
- Re-active
- Re-inactive
 - Njihova uloga je da obezedi funkcionisanje testbenča, odnoso programske podrške u okviru njega ne vezano za izvršavanje dizajna

Gde se šta izvršava



- Na ovaj način je obezbeđen paralelizam rada HW modula koji se simuliraju
- I programa koje pokreće test-benč

Primer rasporedjivanja događaja

```
module race();
    wire p;
    reg q,x,y,z; reg [2:0] a;
    initial begin
        q = 1;
        #1 q = 0;
    end

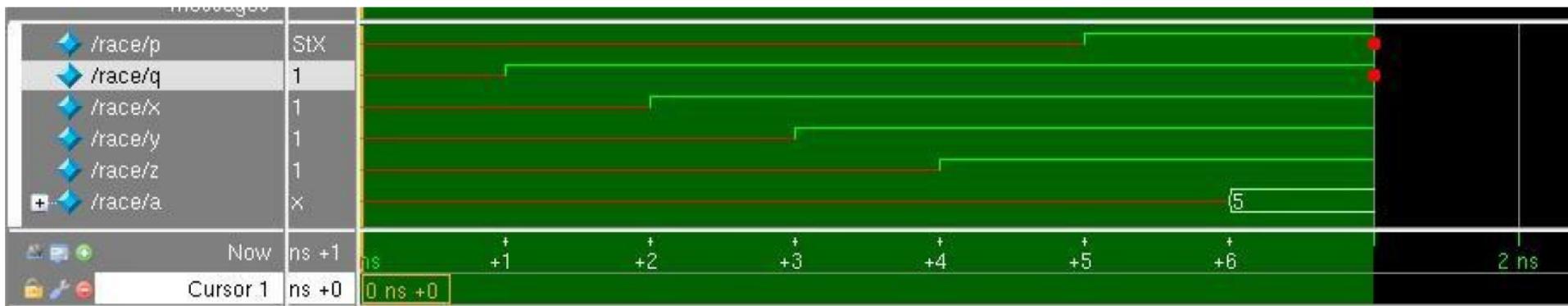
    initial x = 1;
    initial y = 1;
    initial z = 1;

    assign p = q;

    always @(posedge q) a = 5;
endmodule
```

Primer simulacija

- Mentor simulator omogućava ulazak u anatomsiju event-time



Primer opis

- Definisano je (timescale 1ns/1ns)
- Analizirani event time koraci 0ns +0,+1,+2,+3 ...
- Dakle ne govorimo ovde o simulacionim, već o event time koracima; to su “mali” koraci svi unutar 0ns!
- Jedan događaj se postavlja striktno u jedan event time korak - trenutak (Isključivo)
- Pogledajmo redosled: **p** u +5, **q** u +1
- **a** već postavljeno i spremno za događaj #1**q**
- **q** postaje 0 nakon #1, **p** je izmenjeno zbog promene **q**
- Kursor postavljen na 0ns+0
 - **initial blocks** izvršeni u vremenu 0
 - **p** i **a**, neodređeni u vremenu 0

Race između TB i DUT

- Race condition može da se desi između DUT-a i TB-a.
- Ponekad DUT može da bude black box
- TB može da čeka odgovor od DUT-a možda treba da pošalje "potvrdu" u istom simulacionom koraku
- Većina sinhronih DUT radi na ivicu takta
- TB može da se referencira na istu ivicu takta – isti događaj i da uđe u trku sa DUT-om.
- Bolje je ako se može odabrati neki drugi događaj različit od ivice takta na koju radi DUT.
- Signali su stabilni nakon nekog kašnjenja posle aktivne ivice takta.
- Izbegavamo race condition ako TB sempluje signale nakon određenog kašnjenja nakon aktivne ivice.
- Izbegavamo race condition ako TB sempluje signale na suprotnu ivicu takta od DUT-a.

SV Subroutines (Taskovi i Funkcije)

Taskovi

Funkcije

Predavanje argumenata

Statičke i automatske subroutine/variabile

Funkcija nasuprot Taskova

Taskovi

- Taskovi mogu sadržati vremensko kašnjenje (u tom slučaju nisu sintetizibilni)
- Mogu imati prozivoljan broj ulaza i izlaza
- Mogu imati formalne agrumente
- input kopira ulazne vrednosti na počektu taska
- Output kopira izlazne vrednosti na kraju taska
- ref prosleđuje referencu
- Podrazumevani smer je input (ako se ostavi ne sprecificirano)
- Podrazumevani tip argumenta je logic (ako se ostavi ne sprecificirano)

- Taskovi nemaju povratnu vrednost
- Ako se napiše return izraz izaći će se iz taska
- Taskovi mogu da sadrže sekvensijalne i kombinacione blokove
- Mogu da deklarišu lokalne varijable

Primer taska

```
task example_task ( a, b, output c);
#1 c = a + b ;
endtask
```

Default Logic type

Value of
cc ?

```
initial begin
int aa, bb = 3 ; int cc;
example_task (aa,bb,cc);
end
```

Funkcije (C-ovsko nasleđe)

- Vremensko referenciranje i kašnjenje nije dozvoljeno (izvršava se u nultom vremenu - trenutno)
- Sintetizibilne su
- Može imati bilo koji broj ulaza i izlaza
- Vraća jednu vrednost – vektor, real ili integer.
- Mogu da imaju formalne argumente
 - Input kopira vrednost IN na početku
 - Output kopira vrednost OUT na kraju
 - Ref prosleđuje referencu
 - Podrazumevani smer je input (ako se ostavi ne sprecificirano)
 - Podrazumevani tip argumenta je logic (ako se ostavi ne sprecificirano)

- Niz se može specificirati kao formalni argument funkcije
- Pozivanje funkcije sa output, inout ili ref argumentima unutar događaja (event-a) nije dozvoljeno
- Funkcije sa i bez povratne vrednosti (void)
 - U Verilogu, funkcije moraju da vrate vrednost
 - Povratna vrednost se specificira dodelom vrednosti imenu funkcije
 - SV dozvoljava void deklaraciju funkcija
 - Funkcije koje nisu void mogu da vrate vrednost pozivanjem return izraza
 - SV dozvoljava i Verilog tip dodele vrednosti imenu funkcije
- `void'(a_function_with_return_value()); // moguće kastovanje`

```
byte val, clk, temp ;  
....  
function [7:0] add ( byte a , b , c);  
    add = a + b ;  
    $display ("Result ", add); // 5  
endfunction  
  
always @(posedge clk)  
begin  
    val = add(2,3,temp);  
    $display ("Add value is", val); // 5  
end
```

```
byte val, clk, temp ;  
....  
function void add ( byte a , b );  
    val = a + b ;  
endfunction  
  
always @(posedge clk) begin  
    add (2,3);  
    $display (val); // 5  
end
```

```
byte val, clk, temp ;  
....  
function int add (byte a , b );  
    return a + b ;  
endfunction  
  
always @(posedge clk) begin  
    temp = add (2,3);  
    $display (temp); // 5  
end
```

Prosleđivanje argumenata taskovima i funkcijama

- Po vrednosti
 - Podrazumevani način
 - Vrednosti prosleđenje kao argumenti subroutine
 - Promene su lokalne unutar subroutine
 - Alocira se potrebna memorija za svaki poziv subroutine, argumenti se kopiraju svaki put

```
function int add (byte a , b );  
    return a + b ;  
endfunction
```

Prosleđivanje argumenata taskovima i funkcijama

- Prosleđivanje po referenci
 - Argumenti se ne kopiraju subrutini prilikom poziva
 - Referenciraju se na izvornu vrednost argumenta
 - Moraju se poklapati tipovi
 - Kastovanje tipa nije dozvoljeno
 - ref ključna reč mora da se koristi
 - Pri korišćenju ref ključne reči, input, inout, output kvalifikatori se ne koriste
 - Da bi se zaštitio argument prosleđen po referenci, kvalifikator const se može upotrebiti
 - Tada se varijabla može čitati, ali ne i pisati

Prosleđivanje po referenci - primer

```
function automatic int add (ref int x, y);
    return x + y;
endfunction ... ...
temp = add ( xx, yy); // reference, not value
```

function int add (const ref** int x, y);**

Static / automatic subroutines

- Taskovi i funkcije su podrazumevano statičkog tipa u Verilogu-2001.
- Verilog-2001 podržava statik i automatik taskove i funkcije
 - Nema automatic Varijabli
- Static taskovi i funkcije dele isti mem. prostor
 - Unutar instance modula
- Automatik subroutine alociraju jedinstven stek za svaku instancu
- SV podržava
 - Automatic variable unutar statik taskova i funkcija
 - Automatic taskove i funkcije sa static varijablama
 - Automatic taskovi i funkciji podrazumevaju sve interne varijable da budu automatic

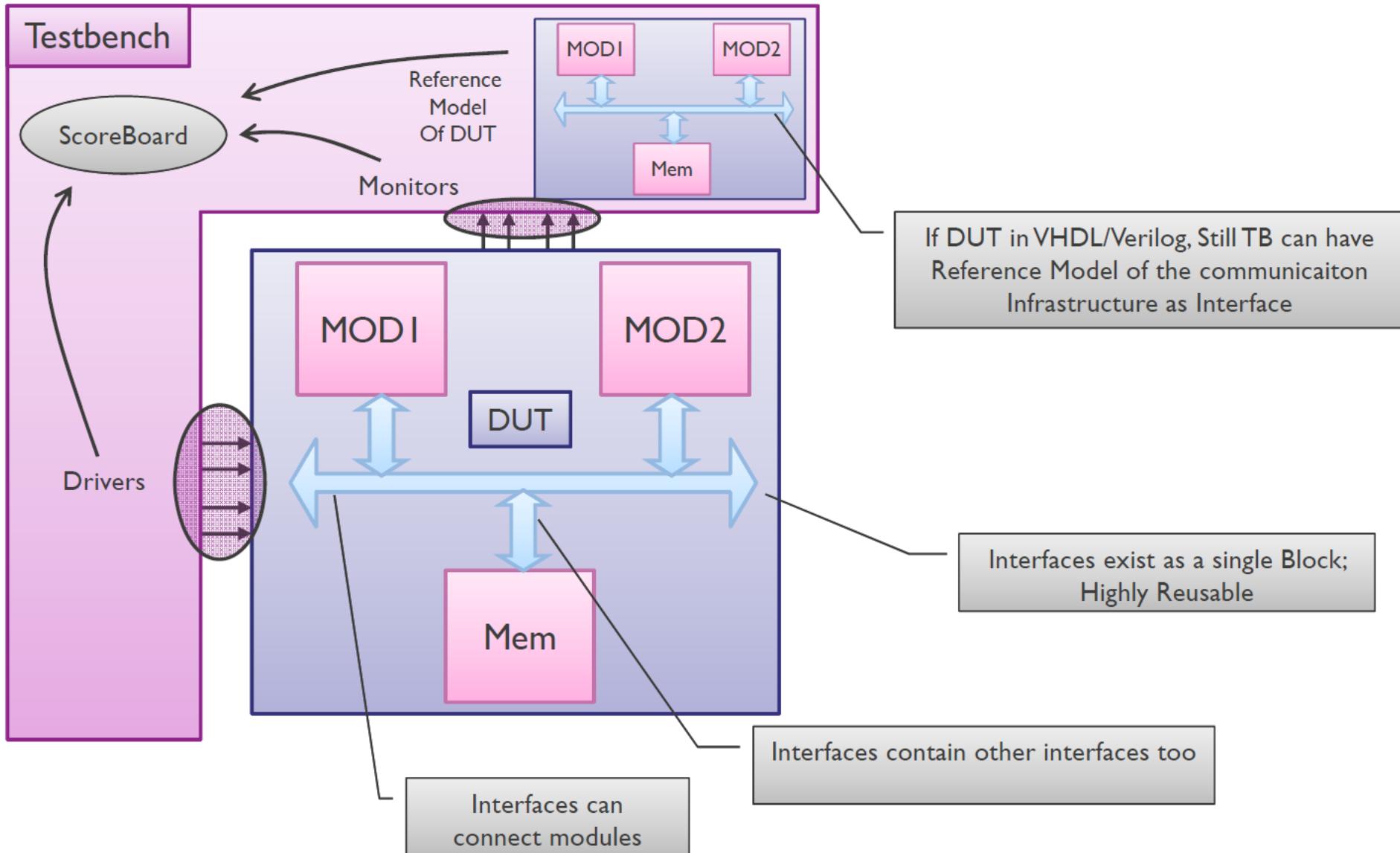
Funkcije napram taskova

funkcije	taskovi
Funkcija ne može da pozove task. Funkcija može da pozove druge funkcije	Task može da pozove funkciju, kao i druge taskove.
Izvršava se u nultom vremenu.	Mogu se izvršiti u nultom vremenu ako ne sadrže vremenska kašnjenja.
Unutar funkcije, ne može se koristiti događaj (event), kašnjenje, ili vremenska kontrola.	Taskovi mogu da sadrže svaki od navedenih vremenskih kontrola ili događaje (event).
Fukcije mogu sadržati argumente tipa output, input, inout ili ref.	Taskovi mogu sadržati argumente tipa output, input, inout ili ref.
Funkcije vraćaju jednu vrednost ili ni jednu ukoliko su deklarisane kao void	Task ne može da vrati vrednost, ali može da prosledi više vrednosti kroz output i inout argumente

Interfejsi

- Omogućavaju efikasno povezivanje između blokova velikih kompleksnih digitalnih sistema
 - Interfejsi,
 - Parametrizovani interfejsi
 - Modportovi
 - Virtualni interfejsi

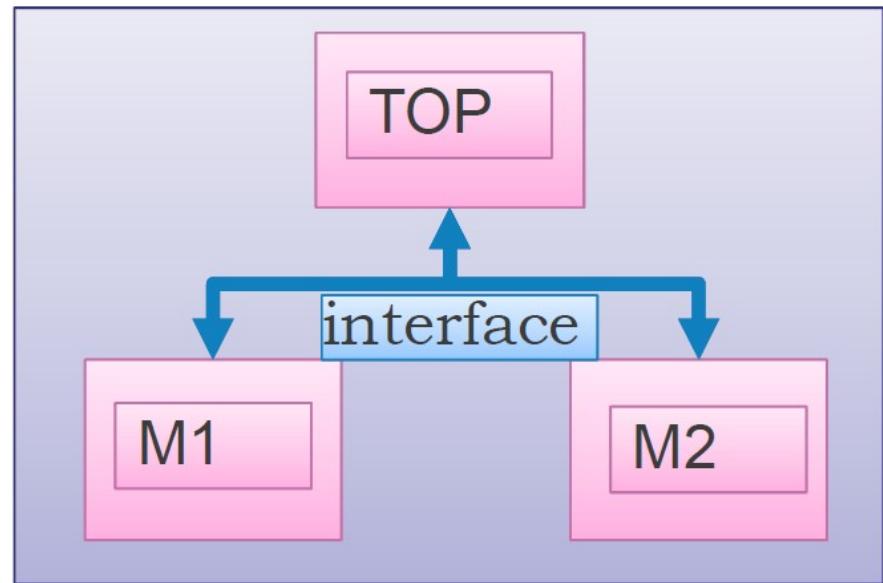
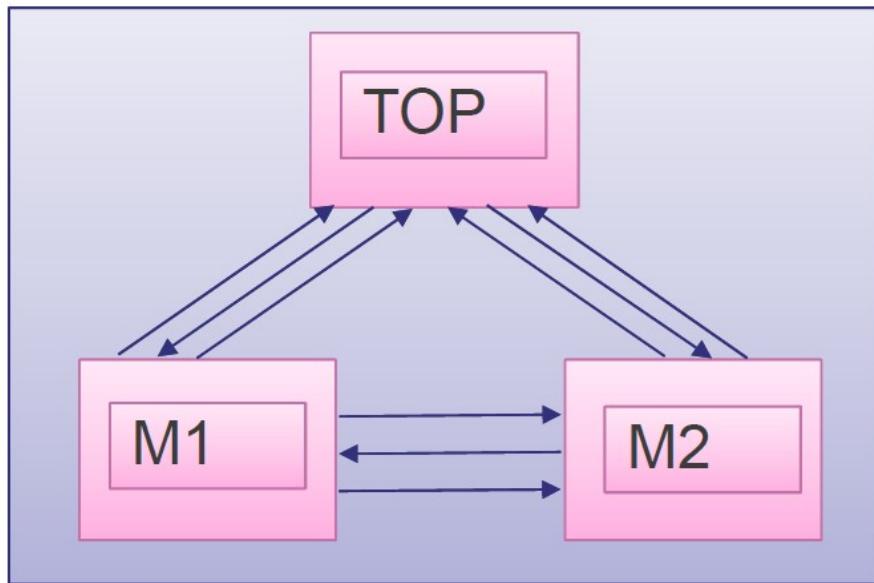
- Komunikacija između blokova digitalnog sistema postaje kritična kako dizajn raste
- Ručno povezivanje stotina portova može lako prouzrokovati greške
- Potrebno je detaljno poznavanje portova
- Teško je menjati i održavati portove u koliko dolazi do promena specifikacija
- Komponente instancirane više puta linearno povećavaju broj konekcija
- Interface konstrukcija u SV je kreirana da bi enkapsulirala komunikaciju između blokva
- Na RTL nivou, unutar interfejsa mogu biti čvorovi ili variable (Sintetizibilno)
- Može se koristiti na različitim nivoima apstrakcije
- Povezuje RTL i sistemski nivo
- Interfejsi može da sadrži takove, funkcije procese, (initial/always) i kombinacione dodele
- Za sistemsko modelovanje i testbenčeve.



Interfejsi

- Ručno povezivanje velikog broja portova često dovodi do grešaka
- Neophodno je detaljno poznavanje svih portova.
- U slučaju promene dizajna, potrebno je ručno prečistiti veliki broj portova retroaktivno
- Višestruko ponavljanje manualnog posla

Bez interfejsa i sa njim



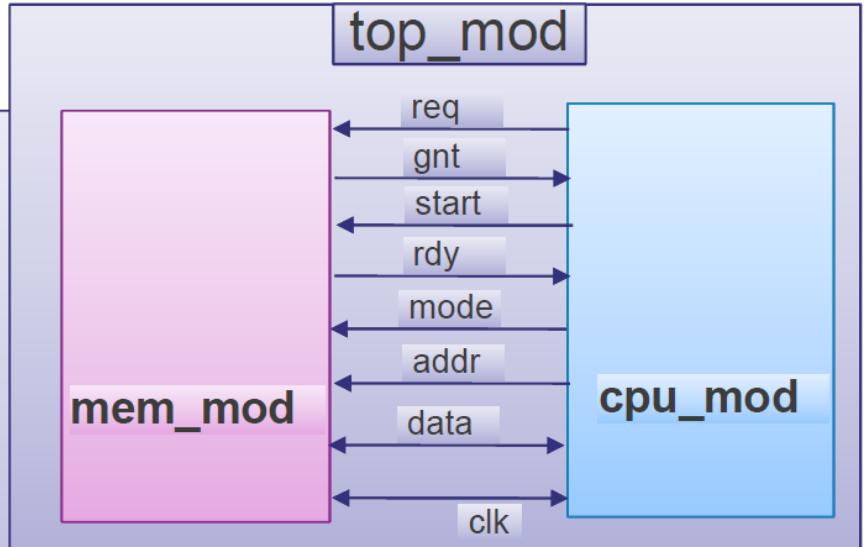
Bez interfejsa

```
module mem_mod ( input bit req,clk,start,  
    logic [1:0] mode,  
    logic [7:0] addr,  
    inout wire [7:0] data,  
    output bit gnt,rdy);  
    logic sel_mem;  
endmodule
```

```
module cpu_mod ( input bit clk,gnt,rdy,  
    inout wire [7:0] data,  
    output bit req,start,  
    logic [1:0] mode,  
    logic [7:0] addr );  
    logic sel_cpu;  
endmodule
```

```
module top_mod;  
  
logic req,gnt,start,rdy;  
logic clk = 0;  
logic [1:0] mode;  
logic [7:0] addr;  
wire [7:0] data;
```

```
mem_mod mem (req,clk,start, mode, addr, data, gnt,rdy);  
cpu_mod cpu (clk,gnt,rdy, data, req,start,mode,addr);  
endmodule
```



Sa interfejsom

```
interface simple_bus;  
    logic req,gnt,start,rdy ;  
    logic [1:0] mode;  
    logic [7:0] addr,data;  
endinterface
```

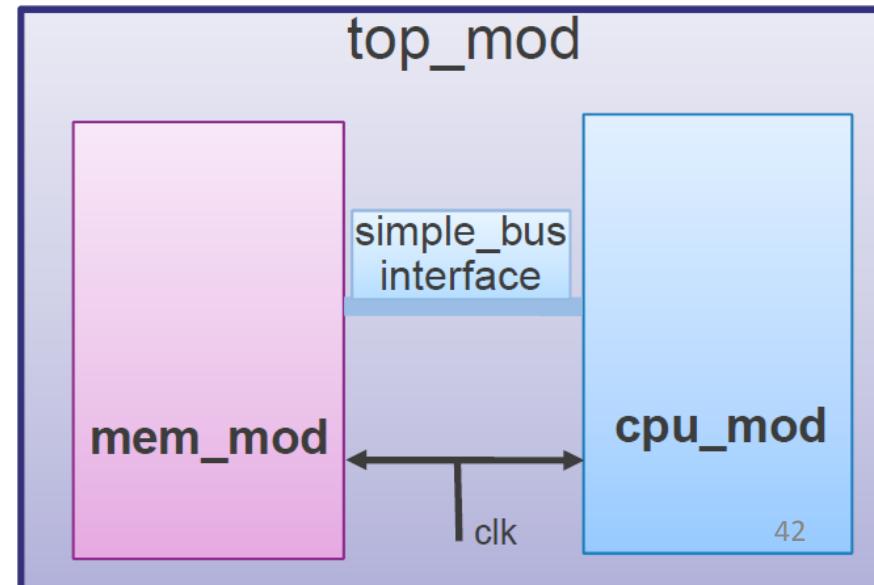
Interface declared here is directionless

Interface members accessed with dot operator

```
module mem_mod ( input bit clk,  
simple_bus busa);  
    logic sel_mem = 1;  
    always @ (posedge clk) begin  
        busa.gnt <= busa.req & sel_mem;  
    end  
endmodule
```

```
module top_mod1;  
    logic clk = 0;  
    simple_bus sb();  
    always #5 clk = ~clk;  
    cpu_mod1 cpu (clk, sb);  
    mem_mod1 mem (clk, sb);  
endmodule
```

```
module cpu_mod ( input bit clk,  
simple_bus busa);  
    logic sel_cpu;  
    always @ (posedge clk) begin  
        busa.mode <= {busa.gnt, busa.gnt};  
        busa.req <= 1 ;  
    end  
endmodule
```



Interfejs priinstanciranju modula

- SV dozvoljava sledeće port maping tehnike
 - Po poziciji
 - Po imenu (.name) sa tačkom kao prefiksom
 - Implicitno mapiranje, ukoliko se sva imena poklapaju (.*)

primer

```
module mod1(input bit clk,  
simple_bus bus1);  
...
```

```
module mod2(input bit clk,  
simple_bus bus2);  
...
```

```
module top_mod1;  
logic clk = 0;  
simple_bus sb1();  
simple_bus bus2();  
  
mod1 a1 (clk, sb1);  
mod1 a2 (.clk(clk), .bus1(sb1));  
mod2 b1 (.clk, .bus2);  
mod2 b2 (.* );  
...
```

Positional reference

Named mapping

Implicit mapping (all
names should match)

Inerfejsi sa genericima

- U headeru modula može se koristiti referenca na nespecificirani interface. To je generički interfejs.
- implicit port (.name) se ne može koristiti da referencira generički interfejs.
 - Samo portovi van interfejsa mogu se implicitno mapirati (.name)
 - Parcijalno Implicitno mapiranja portova je dozvoljeno

primer

```
module mod1(input bit clk, interface bus1);
```

modules can use generic interface port
(Just space allocated for interface)

```
module mod2(input bit clk, interface bus2);
```

Just an interface name is assigned
here for calling it again

```
module top_mod1;  
    logic clk = 0;  
    simple_bus sb1();
```

Now a new Interface instance is
created. Simple_bus should exist !

```
mod1 a1 (.clk(clk), .bus1(sb1));  
mod2 a2 (.clk(clk), .bus2(sb1));  
mod2 a2 (*.); // Error  
mod2 a2 (*.*, bus2(sb1));  
... ...
```

sb1 interface mapped to connect
the a1(mod1) and a2(mod2)

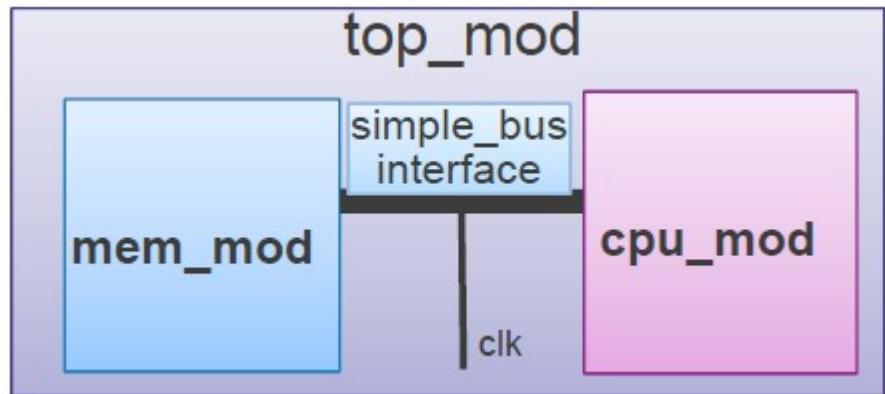
Implicit port mapping is NOT allowed, as it is not clear
which interface the user is referring

Partial Implicit port mapping is allowed

Portovi unutar interfejsa

- Interfejsi mogu imati svoje portove
- Povezivanje slično module portu
- Deli se eksterni signal
- Moduli povezani preko interfejsa mogu da pristupaju članicama datog interfejsa

primer



```
module cpu_mod1 ( interface b );
    logic sel_cpu;
    always @(posedge b.clk) begin
        b.mode <= {b.gnt,b.gnt} ;
        b.req <= 1 ; end
endmodule
```

```
interface simple_bus ( input bit clk );
    logic req,gnt,start,rdy ;
    logic [1:0] mode;
    logic [7:0] addr,data;
endinterface : simple_bus
```

```
module mem_mod1(simple_bus a );
    logic sel_mem = 1;
    always @(posedge a.clk) begin
        a.gnt <= a.req & sel_mem;
    end
endmodule
```

```
module top_mod1;
    logic clk = 0;
    simple_bus bus (clk);
    mem_mod1 mem (.a(bus));
    // cpu_mod1 cpu (*.*) ; // inst name should be b
    cpu_mod1 cpu (bus);
endmodule
```

- Parametrizovani interfejsi
- Slični su parametrizovanim modulima, definišu se generički parametri

```
interface simple_bus #(dwidth =32, awidth = 8) ( input bit clk );
    logic req,gnt,start,rdy ;
    logic [dwidth :0] data;
    logic [awidth:0] addr;
endinterface
```

interface with generic parameters

```
module top ;
    logic clk = 0;
    ...
    simple_bus #(16,4) buss(clk); // dwidth = 16, awidth = 4
    simple_bus #(16) buss(clk); // dwidth = 16, awidth = 8
    simple_bus #( ,4) buss(clk); // Error !!! rather specify #(32,4)
    simple_bus buss(clk); // dwidth = 32, awidth = 8
    ...
endmodule
```

generic parameters passed

Only First value taken, rest skipped

Default generic parameters taken

modport

- Proširuje upotrebu interfejsa unutar modula
 - Specificira smer signala
 - Specificira signale iz interfejsa kojima pristupa modul
 - Zamenjuje port listu

Primer interfejsa sa modportovima

```
module cpu_mod1 ( simple_bus.slave b );
    int yy;
    always @(posedge b.clk) begin
        b.gnt   <= b.start;
        b.req   <= ~b.start ;
        b.addr  <= yy++;  end
    endmodule
```

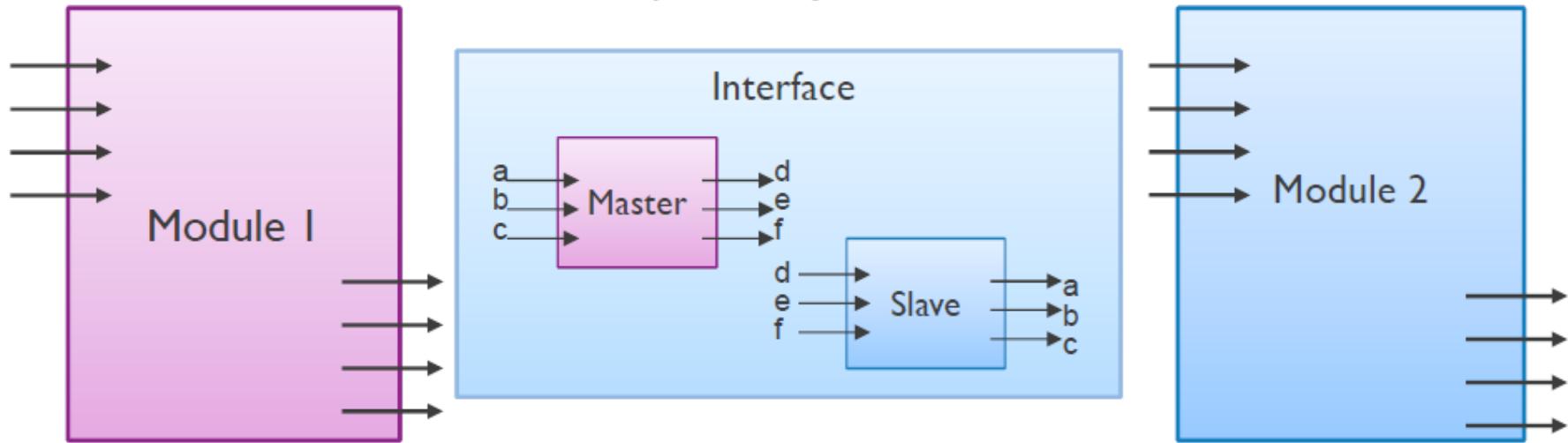
```
interface simple_bus (input bit clk) ;
    logic req,gnt,start,rdy ;
    logic [1:0] mode;
    logic [7:0] addr;
    logic [16:0] data;
    modport master ( input req,gnt,clk,mode,addr,output data, start, rdy);
    modport slave  (output req,gnt,mode,addr,input data, clk, start, rdy);
endinterface
```

```
module mem_mod1 ( simple_bus.master i );
    int cnt;
    always @(posedge i.clk) begin
        i.start   <= i.gnt & i.req;
        i.data    <= cnt++; end
    endmodule
```

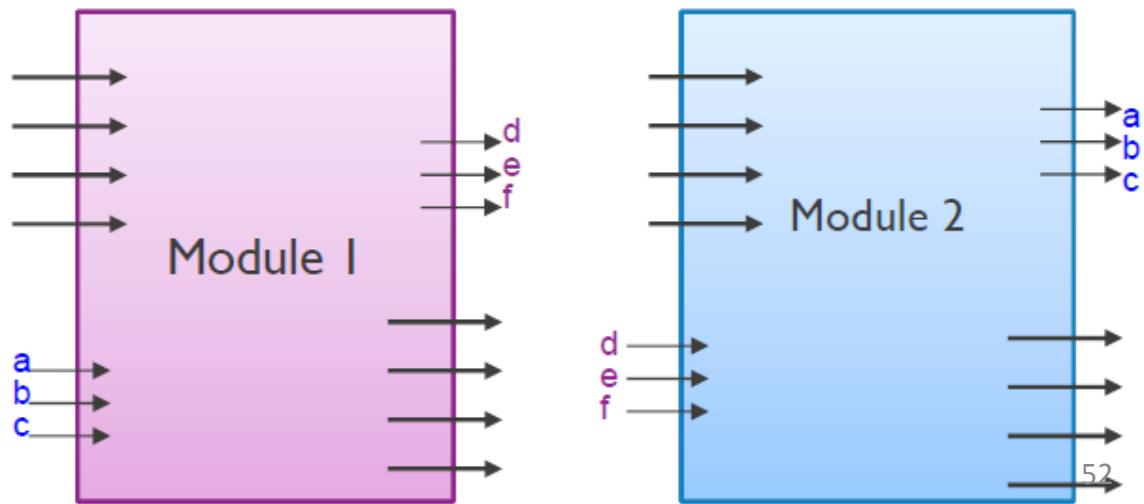
```
module top_mod1;
    logic clk = 0; simple_bus bus(clk);
    always #5 clk = ~clk;
    mem_mod1 mem (.i(bus));
    cpu_mod1 cpu (.b(bus));
endmodule
```

Modportovi

Modport Way



Traditional Way



- simple_bus interfejs definiše master i slave port interfejs
- individualni moduli imaju izbor između master ili slave port interfejsa

```
interface simple_bus (input bit clk) ;
    logic req, gnt, start, rdy ; logic [1:0] mode; logic [7:0] addr; logic [16:0] data;
    modport master ( input req,gnt,mode,addr,output data, start, rdy);
    modport slave (output req,gnt,mode,addr,input data, start, rdy);
endinterface
```

```
module mem (simple_bus.master mbus);
endmodule
```

```
module cpu (simple_bus.slave sbus);
endmodule
```

```
module top;
    simple_bus busa();
    mem mem_i (.mbus (busa));
    cpu  cpu_i  (.sbus (busa));
endmodule
```

- simple_bus interfejs definiše master i slave port interfejse
- Individualnim modulima dodeljen isti interfejs
- ‘top’ modul razrešava master ili slave ulogu instanci

```
interface simple_bus (input bit clk);
    logic req, gnt, start, rdy; logic [1:0] mode; logic [7:0] addr; logic [16:0] data;
    modport master ( input req,gnt,clk,mode,addr,output data, start, rdy);
    modport slave (output req,gnt, mode,addr,input data, start, rdy);
endinterface
```

```
module mem (simple_bus mbus);
endmodule
```

```
module cpu (simple_bus sbus);
endmodule
```

```
module top;
    simple_bus busa();
    mem mem_i (.mbus (busa.master));
    cpu  cpu_i  (.sbus (busa.slave));
endmodule
```

Taskovi i funkcije unutar interfejsa

- Definisanje taskova i funkcija u interfejsima omogućava visok nivo apstraktnog modelovanja
- Čitanje i pisanje može se definisati kao task/funkcija
- import/export taskovi u interfejsima/modportovima

- Taskovi i funkcije unutar interfejsa primer

```
interface simple_buss (input bit clkk) ;  
    logic [4:0] addr; ... ...  
    task mread (input logic [7:0] raddr);  
        addr = raddr[7:0];  
        $display(" Assigning raddr to addr ");  
    endtask  
    task sread ;  
        $display(" I am in SREAD");  
    endtask  
    function int get (int test);  
        get = test ;  
    endfunction  
endinterface
```

```
module mem_mod1 ( interface i ); // Infers any interface  
    always @(posedge i.clkk) begin  
        i.sread ; // I am in SREAD  
        i.mread(44); // Assigning raddr to addr  
        $display("Displaying value", i.get(44)); // 44  
    end  
endmodule
```

```
module top_mod1;  
    logic clkk = 0;  
    simple_buss buss(clkk);  
    always #5 clkk = ~clkk;  
    mem_mod1 mem (.i(buss)); //assign right interface  
endmodule
```

Subrutine u modportovima

- Definisani taskovi i funkcije u okviru interfejsa mogu se importovati u određeni modport
- Čitanje / upis se mogu definisati kao taskovi / funkcije

```
interface simple_bus #(wid1 = 32,wid2 =8) (input bit clk) ;  
    logic req,gnt,start,rdy ;  
    logic [1:0] mode;  
    logic [4:0] addr;  
    logic [5:0] data;  
  
    modport master (input req,gnt,mode,addr ,output data, start,rdy , import task mread(), task mwrite());  
    modport slave (output req,gnt,mode,addr, input data, start,rdy , export task sread(), task swrite());  
  
    task mread (input logic [7:0] raddr);           $display(" I am in MREAD "); endtask  
    task sread ;                                $display(" I am in SREAD "); endtask  
    task mwrite (input logic [7:0] raddr);          $display(" I am in MWRITE "); endtask  
    task swrite ;                               $display(" I am in SWRITE "); endtask  
  
endinterface
```

Virtualni interfejs

- Obezbeđuje mehanizam za razdvajanje apstraktnih modela
- Virtualni interfejsi omogućavaju povezivanje uz funkcionalnu apstrakciju
- Mogu da predstavljaju različite fizičke interfejse
 - Moraju se inicijalizovati pre upotrebe.
 - Mogu se tretirati kao poenteri ka interfejsima

- Virtualni interfejs primer

```
interface sbus ;
    logic req,grant;
    logic [7:0] data, addr;
endinterface

class sbus_c;
    virtual sbus bus; //virtual interface type sbus
    function new (virtual sbus s); bus = s;
    endfunction
    task request(); bus.req = 1 ;
    endtask
    task wait_for_bus(); @(posedge bus.grant);
    endtask
endclass
```

```
module cpu_mod ( sbus s ); endmodule
module mem_mod ( sbus s ); endmodule
```

```
module top_mod;
    sbus s[1:2] (); // instantiate the interfaces
    cpu_mod  cpu  (s[1]); // instantiate 2 modules
    mem_mod  mem  (s[2]); // using sbus interface
    initial begin
        sbus_c tt[1:2]; // create 2 class objects
        tt[1]  = new(s[1]);
        tt[2]  = new(s[2]);
    end
endmodule
```

Prednosti korišćenja interfejsa

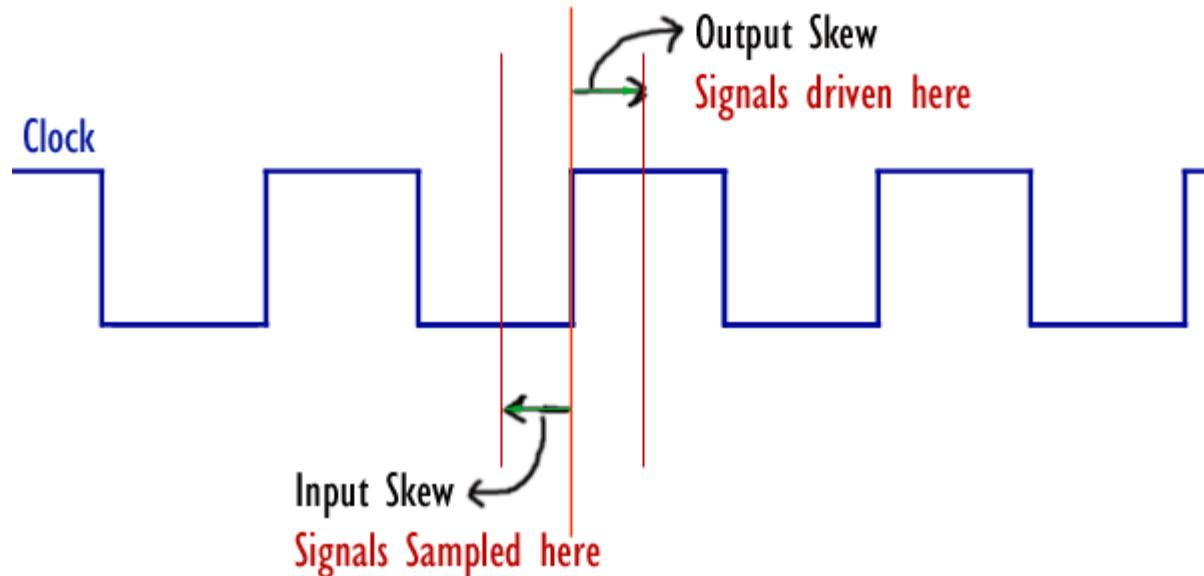
- Interfejs se prosleđuje kao jedinstven objekat
- On kapsulira struktuiran tok podataka/kontrola između blokova
- SV podržava mnoge mehanizme u interfejsima, osim definisanja i instanciranja drugih modula.
 - Povećava reusability
- Interfejs se može definisati u posebnom fajlu
 - Može se posebno i kompajlirati
- Interfejsi mogu da sadrže taskove i funkcije
- Interfejsi mogu da sadrže
 - Protocol čekere; korišćenjem assertiona
 - Proveru funkcionalne pokrivenosti
- Redukuju manualne greške pri ručnom povezivanju

Kloking blok

- Interfejs specificira signale putem kojih blokovi komuniciraju
- Međutim, interfejs ne specificira vremenske karakteristike, sinhronizaciju, niti takt
- SV za tu svrhu uvodi kloking blok
 - Definiše takt signale
 - Definiše tajming i sinhronizacione zahteve

Kloking blok primer

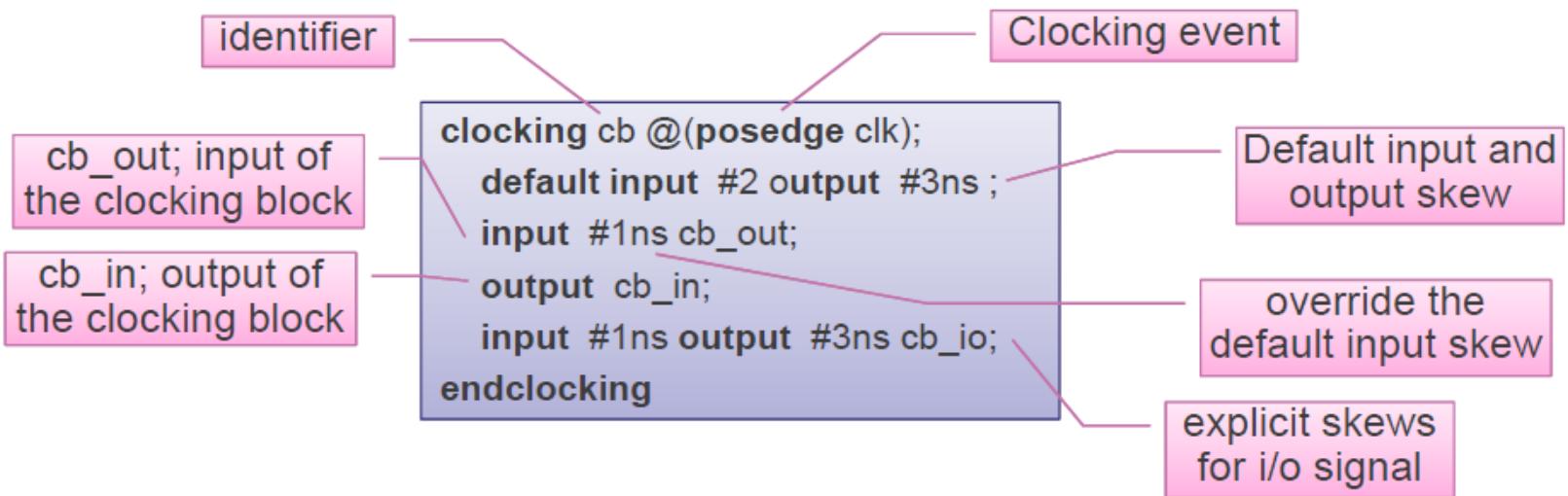
```
clocking cb @ (posedge clk);
  default input #1 output #2;
  input from_Dut;
  output to_Dut;
endclocking
```



- input #1 // opredeljuje trenutak semplovanja ulaznog signala u odnosu na referentnu ivicu takta (u ovom slučaju to je 1 time unit), pre ivice za 1
- output #2 // opredeljuje trenutak postavljanja izlaznog isgnala u odnosu na referentnu ivicu takta (u ovom slučaju to je 2 time unita), posle ivice za 2
- Pošto opisuje odstupanje od referentne ivice takta, ime mu je skew (nesigurnost, smicanje)

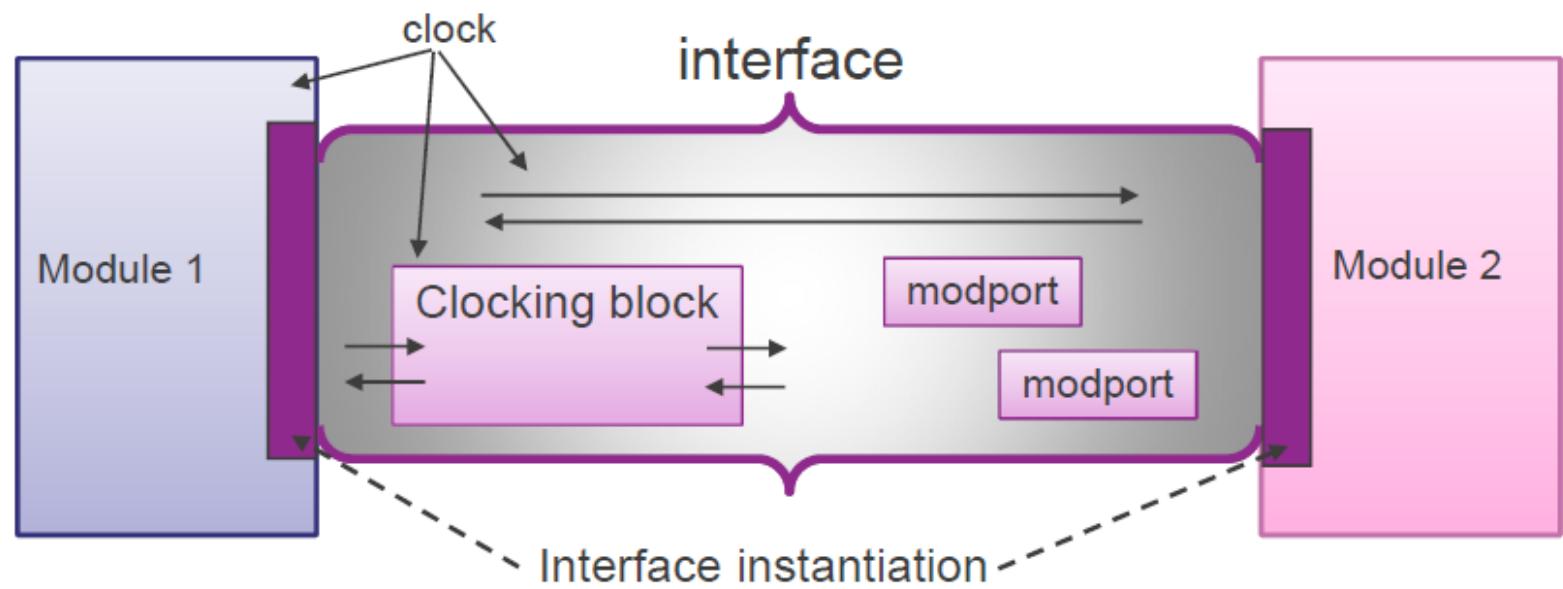
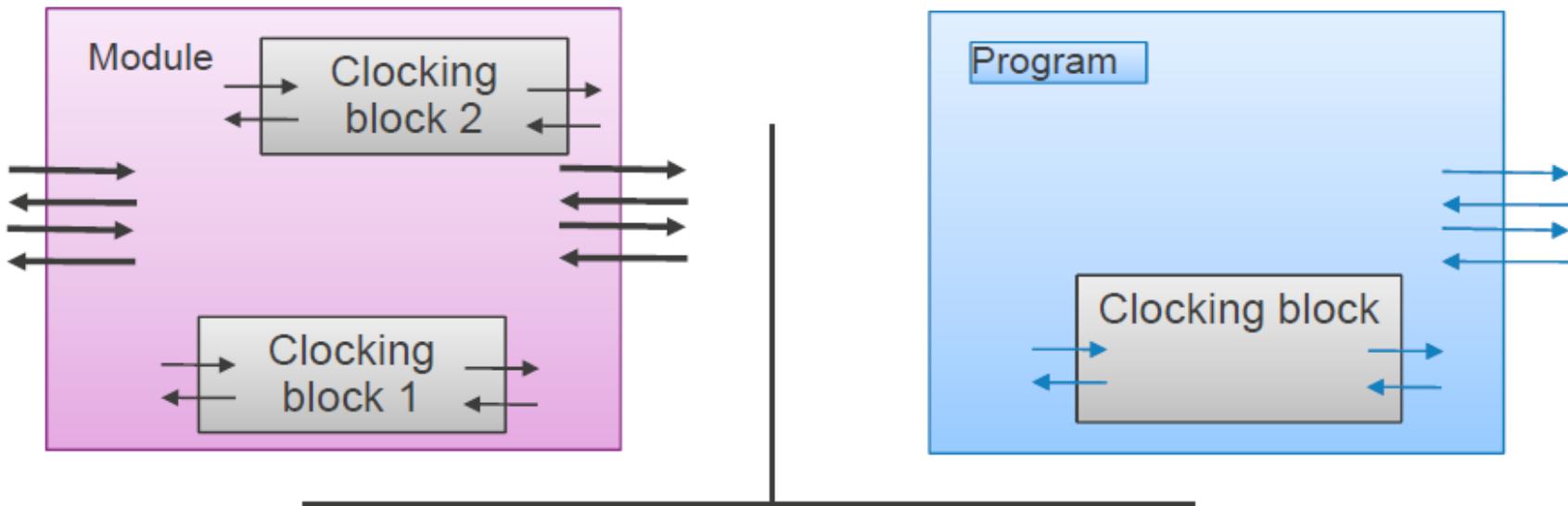
- Kloking blok dodaje sampling i driving tajming signalima
 - Ne deklariše signale
 - Definiše tajming u odnosu na dati takt
- Pomaže da se izbegne race (trka) između takta i podataka
- Kloking blok se može instancirati u okviru:
 - modula/ programa/ interfejsa

- Skew specificira kada je signal na ulazu/izlazu semplovan pre/posle kloking događaja
- Skew mora biti konstantan izraz, može se specificirati i kao parametar.
- Može se unosti u:
 - #1; // time unit koracima podrazumevano
 - #2ns ; //vremenskim jedinicama
 - ##4; //ciklusima takta



- Kloking blokovi se ne mogu ugnježdavati
- Više kloking blokova mogu da koegzistiraju
 - Unutar različitih interfejsa
 - Više takt domena
 - U grupisanim signalima izvedenim iz kloka
- Ključna reč default
 - Unutar kloking bloka default specificira skew
 - Izvan kloking bloka default specificira sam blok
 - Samo jedan default blok može da postoji unutar modula/interfejsa/programa

```
module top;  
    bit clk;  
    logic x1,x2,x3,x4,x5,x6;  
    clocking cb1 @(posedge clk);  
        default input #1ns output #2ns;  
        input x1;  
        output x2;  
    endclocking  
    clocking cb2 @(negedge clk);  
        input #1ns x3;  
        output #3 x4;  
    endclocking  
    default clocking cb3 @(posedge clk);  
        default input #1ns output #2ns;  
        input x5;  
        output x6;  
    endclocking  
    // default clocking cb3;  
endmodule
```

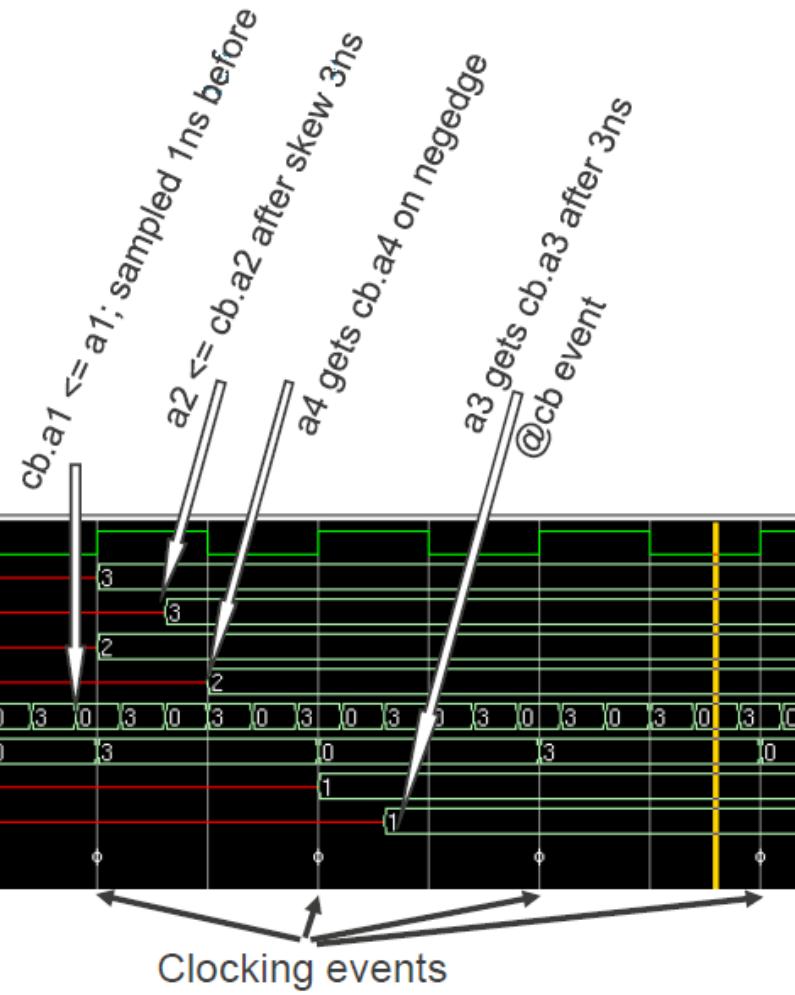


```

module top;
  logic [2:0] a2, a3, a4; bit [1:0] a1;
  initial for( int i =0;i<20;i++) clk = #5 ~clk;
  initial for( int i =0;i<20;i++) a1 = #2 ~a1;
  clocking cb @(posedge clk);
    default input #1ns output #3;
    input a1 ;
    output a2 , a3 ;
    output negedge a4;
  endclocking
  initial begin
    @(cb);
    cb.a2 <= 3'b11;
    cb.a4 <= 3'b10;
    @(cb);
    cb.a3 <= 3'b01;
  end
endmodule

```

◆ /top/clk	0
+◆ /top/cb/a2	3
+◆ /top/a2	3
+◆ /top/cb/a4	2
+◆ /top/a4	2
+◆ /top/a1	0
+◆ /top/cb/a1	3
+◆ /top/cb/a3	1
+◆ /top/a3	1
◆ /top/cb_cb_event	



- @cb refers to the event sensitivity specified in cb
- @cb.signal waits until next event on signal #1step before sensitivity event

Kloking blokovi pregled

- Osnovne k-ke kloking blokova
 - Semplovanje ulaza
 - Sinhrono drajvanje
 - Sinhronizacija događaja
- Skew se može specificirati na 3 načina
 - #T : skew izražen u T vremenskim jedinicama. Njih opredeljuje parametar simulacije timescale
 - #Tns : skew izražen u T nano sekundi (u vremenskim jedinicama).
 - #1step : semplovanje se obavlja u preponed preponed regionu datok vremenskog koraka. Podsetiti se priče o SV raspoređivaču
- ## vrednost kašnjeja kao broj takt ciklusa
 - ##3 ; //cekaj 3 ciklusa takta
- Kloking blokovi se ne mogu deklarisati u subrutinama ili packagima
- Kloking blokovi se mogu deklarisati u modulima/interfejsima ili programima
- Kloking blok događaj @(cb)
- Ima svoje podrazumevane input i output skew

Nizovi i redovi izvršenja (queue)

- Niz je kolekcija varijabli istog tipa podataka
- Verilog 2001, dozoljava nizove svih tipova podataka
- SV povrh svega dodaje pakovane (packed) i multidimenzione nizove
- Pakovani nizovi
 - Multidimensioni niz može da se tretira kao jedinstven vektor.
 - Može da se sastoji iz single bit tipova (bit, logic, reg, wire)
 - Elementi niza se smeštaju u memoriju kontinualno.
 - Može se čitati/pisati bilo koji element ili bilo koji segment iz niza
- Unpacked nizovi
 - Mogu ih sačinjavati bilo koji tipovi podataka.
 - Elementi niza su pakovani posebno, pod kontrolom simulatora
- Maksimalna veličina pakovanog niza je **2^16** bita.
- Kod pakovanih nizova dimenzija je deklarisana pre imena
bit [3:0] array_p;
- Unpacked dimenzija je deklarisana nakon imena
real array_un_p [3:0];

Mešani nizovi

- SV unpacked nizovi su zapravo tradicionalni nizovi
- Unpacked nizovi mogu biti bilo kog tipa.
- Pakovani nizovi su podrazumevano neoznačeni, dok su svi “integer” tipovi označeni.
- Mešanje pakovanih i unpacked je dozvoljeno.
 - bit a1[7:0] ; // *unpacked array of bit* ; a1
 - bit [7:0] a2 [3:0] ; // a2 is 1-D Unpacked array of 1-D Packed array.
 - bit [7:0][4:0] a3 [3:0] ; // a3 is 1-D Unpacked array of 2-D Packed array.
 - bit [7:0][4:0] a4 [3:0][2:0] // a4 is 2-D Unpacked array of 2-D Packed array.
 - bit [7:0][4:0][3:0][2:0] a5 // a5 is 4-D Packed array.
- Kopiranje pakovanih nizova se tretira kao kopiranje jednog vektora
- Moguće je mešanje ali ćemo ga izbeći u okviru ovog kursa

Pakovani naspram unpacked nizova

```
initial begin // Packed Example
    bit [1:0][4:0] data ;
    byte bye;
    data[1][3:2] = '1;
    bye = data[1];
    $display("displaying by %p and data %p ",bye, data);
end
```

displaying by 12 and data '{12, 0}

```
initial begin // Un-Packed Example
    bit data [1:0][4:0] ;
    byte bye;
    data[1][1] = '1; // data slicing NOT allowed in unpacked
    //bye = data[1]; // Cannot assign an unpacked type to a packed type
    $display("displaying by %p and data %p ",bye, data);
end
```

displaying by 0 and data '{'0, 0, 0, 1, 0}, '{0, 0, 0, 0, 0}'}

Dinamički nizovi

- Dinamički niz je jednodimenzioni unpacked niz čija veličina može biti menjana tokom izvršavanja simulacije
 - Sintaksa: **data_type array_name [];**
 - Primer logic [3:0] test_array[];
//dinamički niz 4bit vektora
 - Dinamičke nizove generišemo operatorom **new[]**
alocira se memorijski prostor i inicijalizuju se članovi niza
 - Veličinu proveravamo funkcijom **size()**,
 - Brišemo ih funkcijom **delete()**, briše sve članove niza i postavlja veličinu na 0.

Primer dinamičkog niza

```
integer i, mem [ ]; // Dynamic array mem
mem = new [5] ;//creates an array of size 5 (0 to 4)
$display ("= %d",mem.size()); //5
for (i=0;i<5;i++) mem[i] = i ;
mem = new [10] (mem); //extends existing array
for (i=5;i<10;i++) mem[i] = i ;
$display ("%d",mem.size()); //10
mem.delete(); //clear all the array and set size to 0
$display ("%d",mem.size()); // 0
```

Funkcije za analizu nizova

- $\$left$ vraća levu granicu dimenzije niza (msb)
- $\$right$ vraća desnu granicu dimenzije niza (lsb)
- $\$low$ vraća minimum između $\$left$ i $\$right$
- $\$high$ vraća maksimum između $\$left$ i $\$right$
- $\$increment$ vraća 1 ako je $\$left$ veći ili jednak $\$right$, a -1 ako je $\$left$ manji od $\$right$
- $\$size$ vraća broj elemenata u dimenziji, koji je jednak $\$high - \$low + 1$
- $\$dimensions$ vraća broj dimenzija u nizu

Metode za manipulaciju sa nizovima

- Lokator metode: obezveđuju pretragu za elementima niza ili indeksima.
- Metode za sortiranje niza
- Metode za redukciju nizova
- Metode za pretragu indeksa

Lokator metode

- `find()` vraća sve elemente koji zadovoljavaju dati izraz
- `find_index()` vraća indekse svih elemenata koji zadovoljavaju dati izraz
- `find_first()` vraća prvi element koji zadovoljava dati izraz
- `find_first_index()` vraća indeks prvog elementa koji zadovoljava dati izraz
- `find_last()` vraća poslednji element koji zadovoljava dati izraz
- `find_last_index()` vraća indeks poslednjeg elementa koji zadovoljava dati izraz
- `min()` vraća element minimalne vrednosti, ili čija vrednost u datom izrazu daje minimum
- `max()` vraća element maksimalne vrednosti, ili čija vrednost u datom izrazu daje maksimum
- `unique()` vraća sve jedinstvene elemente niza (ponovljenje elemente uklanja)
- `unique_index()` vraća indekse svih jedinstvenih elemente niza (ponovljenje elemente uklanja)

- Metode za sortiranje
 - Sortiranje elemenata 1D niza ili reda izvršenja (Queue)
 - **function void ordering_method (array_type iterator = item)**
 - `reverse(); sort(); rsort(); shuffle();`
- Metode redukcije niza
 - Redukuju niz na jednu vrednost
 - **function expression_or_array_type reduction_method**
 - `sum(); product(); bit-wise and(); bit-wise or(); logical xor();`
- Iterator index querying
 - Vraća vrednost indeksa specificirane dimenzije
 - **function int_or_index_type index (int dimension = 1)**

Asocijativni nizovi

- Dinamički nizovi su korisni ako
 - su varijable podataka kontinualne i njihovi indeksi se menjaju dinamički.
- Asocijativni nizovi su korisni ako
 - Prostor podataka nije ograničen ili je sporadično popunjeno
- Kod asocijativnih nizova, indeks niza pored integera može biti bilo kog tipa.
 - `data_type array_name [index_type] ;`
 - `logic [7:0] addr_array [string]; //associative array indexed by string`
- Asocijativni nizovi imaju sledeće metode
 - `num()`, `delete()`, `exists()`, `first()`, `last()`, `next()`, `prev()` + standardne metode manipulacije s nizovima
- Asocijativni nizovi se mogu dodeljivati samo drugim asocijativnim nizovima sa indeksima istog tipa
 - `integer addr_array [string] = {"str1":22, "str2":33, default:44 };`

- Asocijativni niz ne zahvata memorijski prostor dok se ne upotrebi
 - Indeks može biti bilo koji packed tip, string ili klasa
- Asocijativni elementi se skladište u specifičnom redosledu radi bržeg pristupa
- Asocijativni niz implementira lukap tabelu za pristup elementima po tipu
 - `data_type array_id [index_type];`
 - `integer int_Array[*]; // Integer asocijativni niz bez specificiranog indeksa`
 - `logic [7:0] assoc [int]; // Asocijativni logic niz [7:0] sa integer indeksom`
 - `int arr_name [string]; // index string`
- Elementi u asocijativnom nizu su alocirani dinamički
- Elementi u asocijativnom nizu su unpacked

- Indeks može biti džoker
 - Int array_name[*];
 - Indeks može biti bilo koji tip podatka
- String kao indeks
 - int array_name[string];
- Klasa kao indeks
 - int array_name[class_instance];
- Integer ili int indeks
 - logic[2:0] arr_name[int]
- Označeni packed array
 - typedef bit signed [3:0] half_byte;
 - int arr_name[half_byte];
- Neoznačeni packed array ili packed struct
 - typedef bit unsigned [3:0] half_byte;
 - int arr_name[half_byte];

Redovi izvršenja (queue)

- Queue je kolekcija složenih homogenih elemenata
- Queue se deklariše znakom \$ umesto specifikacije integralne veličine ili tipa (asocijativni niz).
- Queue podržava pristup svim elementima i ubacivanje, izbacivanje elemenata na početku/kraju
 - integer integer_queue[\$] ; //veličina nije specificirana
 - logic [7:0] addr_queue[\$:100]; // max veličina queue je 100
- Queue može da se koristi kao FIFO (ima širu funkcionalnost nego FIFO)
 - PUSH, POP, INSERT, REMOVE, SIZE
- Za razliku od asocijativnih/dinamičkih nizova ne može se izbrisati ceo queue.
- Queue metode prdržavaju
 - Ubacivanje, ekstrakciju i izbacivanje elemenata.
- Queues se mogu kopirati ako tip pdatka i veličina odgovaraju.
- Queues se mogu parcijalno seći i kopirati

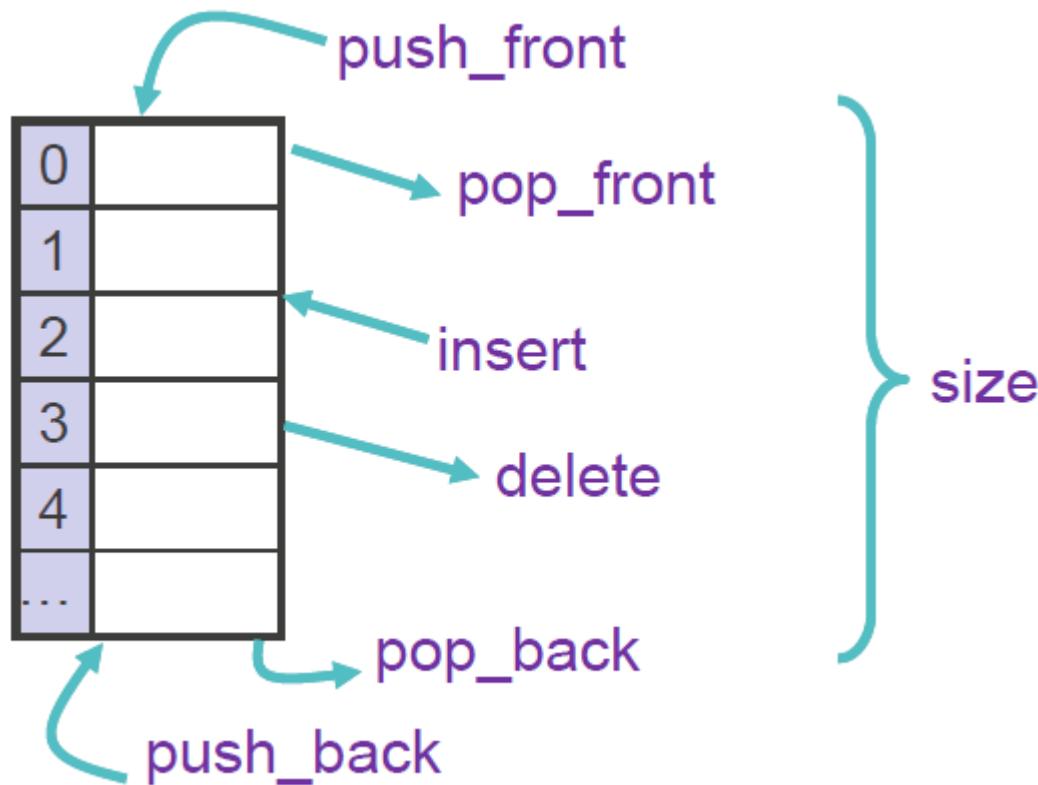
Metode redova izvršenja

size ()	function int size(); <i>Vraća broj elemenata u redu izvršenja</i>
insert()	function void insert (int index, queue_type item) <i>Ubacuje element na specificiranu index poziciju</i>
delete()	function void delete (int index); <i>Briše element sa specificirane index pozicije.</i>
pop_front()	function queue_type pop_front(); <i>Sklanje i vraća prvi element iz reda izvršenja.</i>
pop_back()	function queue_type pop_back(); <i>Sklanja i vraća poslednji element iz reda izvršenja</i>
push_front()	function void push_front (queue_type item); <i>Ubacuje dati elemenat na početak reda izvršenja.</i>
push_back()	function void push_back (queue_type item); <i>Ubacuje dati elemenat na kraj reda izvršenja.</i>

primer

```
logic [9:0] a1 [$:4] ; integer q1 [$];
for (int i=110 ; i <=114; i++) a1.push_front(i); // a1 = {114 113 112 111 110}
for (int i=0 ; i <=1; i++) q1[i] = a1.pop_front(); // q1={114 113} a1={112 111 110}
q1.push_back(5); // q1 = {114 113 5}
q1.insert(2,4); // q1 = {114 113 4 5 }
q1.delete(1); // q1 = {114 4 5 }
q1.push_back(25); // q1 = {114 4 5 25 }
q1.push_back(252); // q1 = {114 4 5 25 252}
q1.delete(1); // q1 = {114 5 25 252}
q1.delete(1); // q1 = {114 25 252 }
q1.insert(2,14); // q1 = {114 25 14 252 }
q1.insert(1,33); // q1 = {114 33 25 14 252}
q1.insert(1,56); // q1 = {114 56 33 25 14 252}// last entry lost
size_q1 = q1.size(); // size_q1 = 6
size_a1 = a1.size(); // size_a1 = 3
a1.delete(0); a1.delete(0); a1.delete(0);
// a1 queue is empty; q1={114 56 33 25 14 252}
for (int i=0; i < 5; i++) a1[i] = q1.pop_front(); // a1= {114 56 33 25 14}
```

ilustracija



pregled

Statički niz

Veličina mora biti poznata pre kompjuiranja

Kratko vreme pristupa svakom elementu

Ako aplikacija ne koristi sve elemente memorija se troši uzaludno

Nije dobro kada se veličina menja

Dobro je kada su podaci kontinualni

Dinamički niz

Veličina nije potrebna u toku kompjuiranja

Veličina se opredeljuje/menja tokom izvršenja programa

Performanse pristupa elementima su iste kao kod statičkog niza.

Memorijska alokacija je povoljna, jer se veličina menja dinamički.

Asocijativni niz

Veličina nije potrebna u toku kompjuiranja.

Vreme pristupa elementu raste sa povećanjem niza.

Malo memorijsko opterećenje ukoliko su nizovi skokoviti.

Korisnik ne mora da brine o veličini niza (automatic resizing)

Redovi izvršenja Queues

Veličina nije potrebna u toku kompjuiranja.

Performanse pristupa elementima su iste kako kod statičkog niza.

Korisnik ne mora da brine o veličini niza (automatic resizing)

FIFO ili LIFO se lako modeluju

Objektno orjentisana verifikacija

- Šta je OOP
- Zašto OOP
- Odakle dolazi OOP
- Zašto ga koristimo u funkcionalnoj verifikaciji hardvera
- Integracija OOP u SV
- Zašto SW koncepti u HW

Evolucija OOP

- Programiranje podrazumeva transformaciju specifikacija/zahteva u nešto što računar može da izvrši
- Evolucija
 - Mašinski jezik
 - Asembler
 - Proceduralni jezici (C, Fortran, ...)
 - OOP
- Samo jedna stvar se nije promenila tokom celokupne evolucije
 - !!!!! PROMENA !!!!!
- Specifikacije/ Zahtevi se stalno menjaju
- Stoga se kod menja, verifikacija se menja
- OOP svodi zahteve na
 - Objekte i njihove okvire
- Minimizovati promene u kodu prilikom promene zahteva!
- Structuralna Arhitektura dozvoljava reuse koda

Osnovni OOP koncepti

- Klase;
 - Prethodnik naslednik, nasleđivanje
- Očuvava apstrakciju
- Klasa je odgovorna za svoj deo posla
- Komunikacija sa kim je potrebno (zaštićeno, privatno, ...)
- Enkapsulirano
- Program razgovara sa interfejsima, ne sa implementacijom

Zašto OOP u HW verifikaciji

- Sve se može uraditi i u proceduralnim jezicima, zašto OOP ?
 - Veća produktivnost !
- Prikrivanje podataka i apstrakcija podataka
 - Način organizacije velike količine koda u prihvatljive delove
- modifikacije/promene; samo nad specifičnim klasama
 - Ostatak okruženja ostaje isti!
- Virtualizacija; pomaže pri kreiranju koda sa sistemskog nivoa ka RTL
 - Virtualne klase i metode, definiše strukturalni prototip, ali bez implementacije
- za test-benčeve, klase može da sadrži podatke i metode zajedno
 - Driver (šalje telegrame), Monitor prati, ...
- Može da koristi na pin nivou signalni nivo u okviru transakcije (apstrahuje)
- Koristi biblioteke u razvoju, kao osnovne klase i omogućava njihovo proširenje tokom razvoja
- Štedi vreme

Proceduralno nasuprot OOP

```
struct driver { logic enable, rd, wr };  
void send (driver d, data, addr);  
void receive (driver d);  
void buffer (driver d, count);
```

Procedural

```
driver mydrive;  
begin  
    send (mydrive,data,addr);  
    receive (mydrive);  
    buffer ( mydrive, count);  
end
```

U proceduralnom pristupu funkcije i varijable ostaju otvorene, svako može da ih modifikuje

```
struct driver { // or class  
    logic enable, rd, wr;  
    void send (data, addr);  
    void receive ();  
    void buffer (count);  
};
```

Data Abstraction

```
driver mydrive;  
begin  
    mydrive.send (data,addr);  
    mydrive.receive ();  
    mydrive.buffer (count);  
end
```

u OOP pristupu funkcije i varijable ostaju kapsulirane

- Dizajner ne može da barata sa brojnim detaljima, kako složenost projekata raste
 - Rešenje: Ne u povećavanju broja inženjere na projektu, nego u povećanju apstrakcije, reuse (ovo je mišljenje iz ugla menadžmenta...)
 - Prikupiti detalje i apstrahovati ih
 - Susedni blok, ne mora da zna interne detalje mog bloka
- Lako debagovanje, baratanje sa greškama
- Korišćenjem UML dijagrama, čak i menadžment može da razume dizajn
- Čist, jednostavan, efikasan dizajn
- Dizajneri imaju više vremena da misle o specifičnom problemu
 - Strukturizacija problema
 - Mora se pisati čist kod
- Koriste se odnosi, nasleđivanje, reference...
- Ne možemo pobeći, uskoro ćemo svi morati da radimo u OOP okruženju

Modelovanje nivoa transakcije

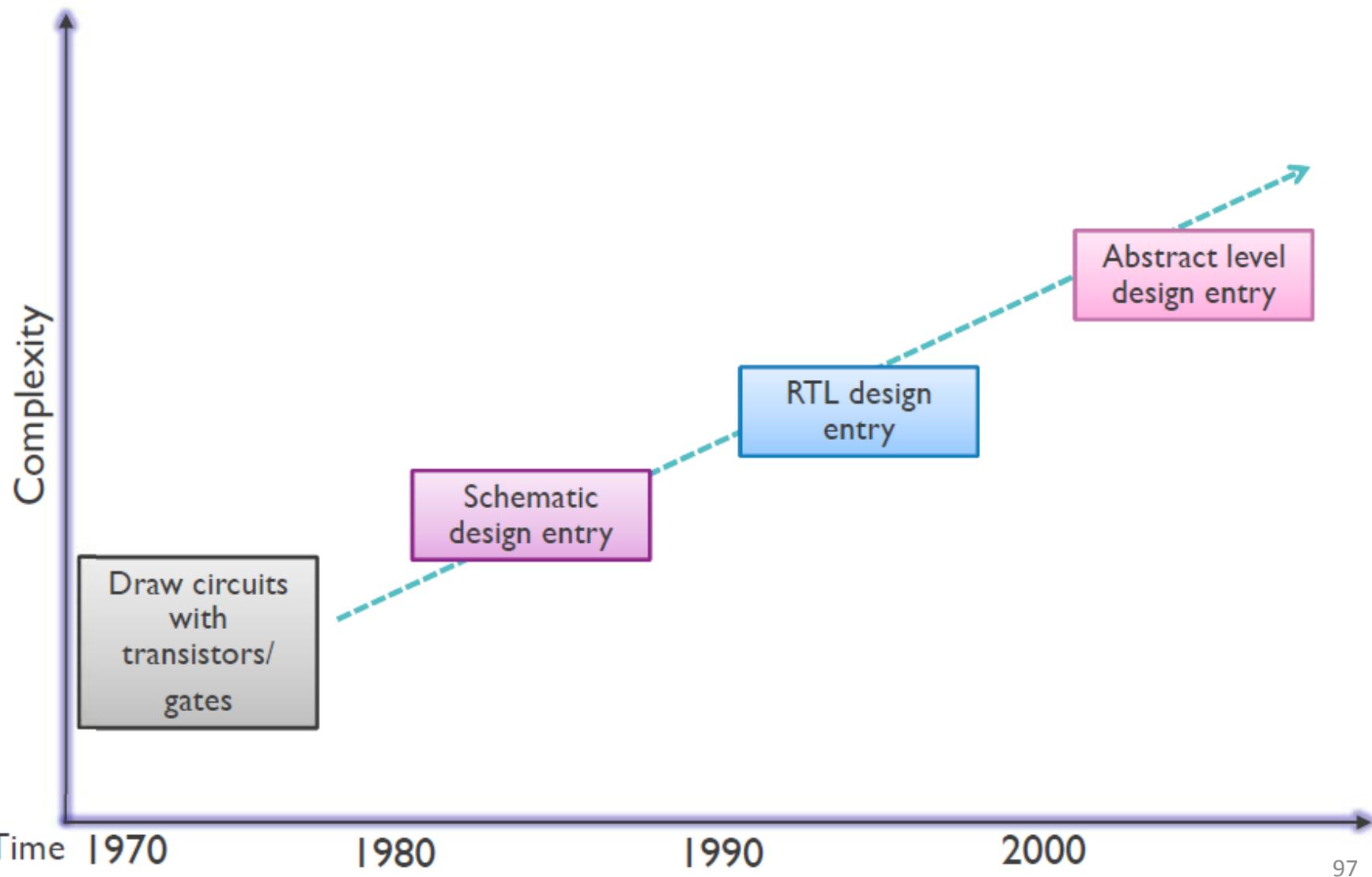
TLM (Transaction Level Modeling)

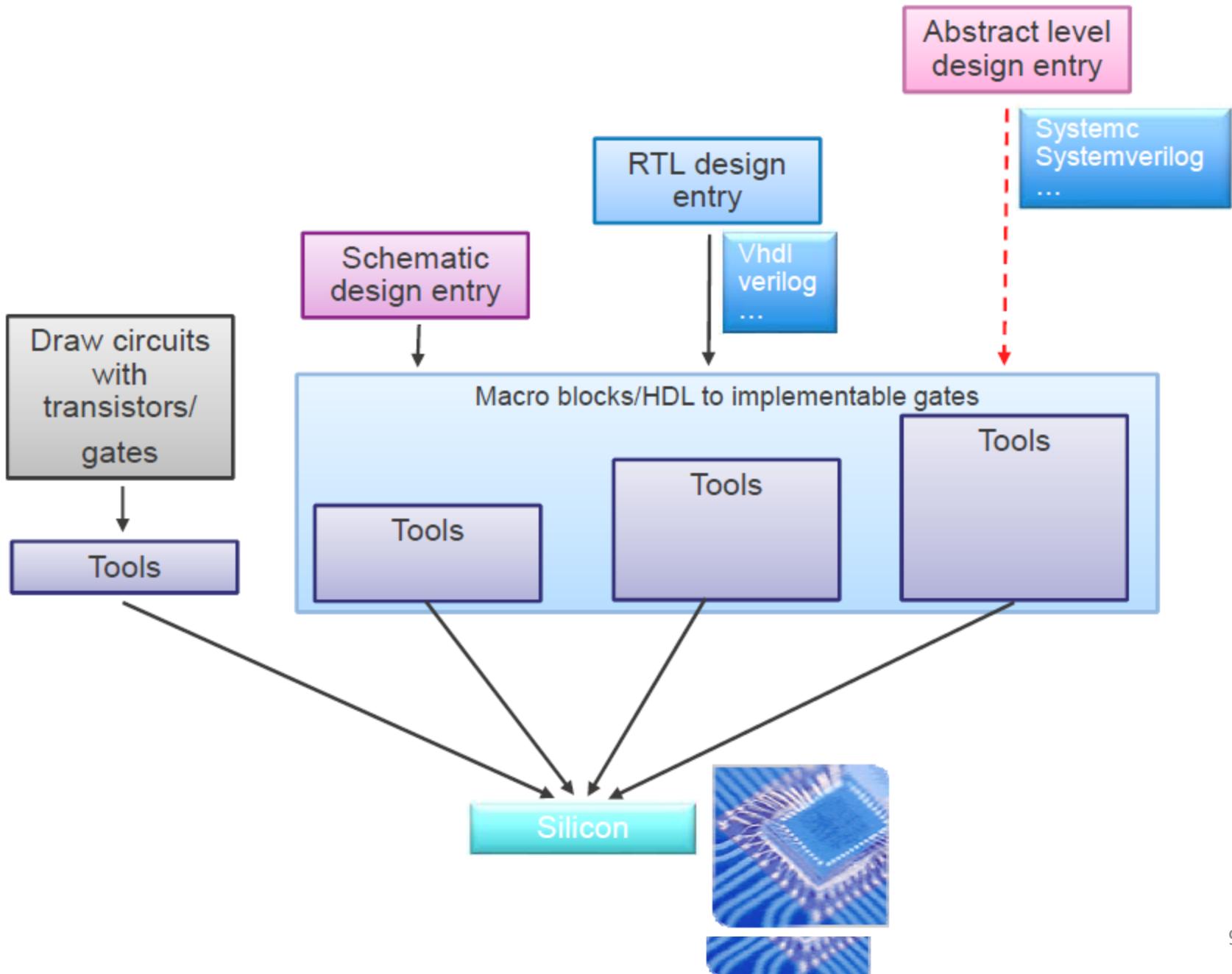
- Transakcija predstavlja jedan transfer podataka ili kontrole između transaktora i dizajna preko interfejsa
 - mem_read () ; bus_write ();
 - Transfer struktuiranog paketa
- Verifikacija bazirana na Transakciji povećava produktivnost, ubrzava simulaciju.
 - reusable
- Omogućava samopроверавајући parametrizovani testbenč na sistemskom nivou
- Prikazuje / analizira odgovore na sistemskom / nivou transakcije kao dodatak na signalni nivo
- Debugovanje olakšano
- Omogućava funkcionalnu pokrivenost dizajna
- Usmereno slučajno testiranje sa ispitivanjem graničnih slučajeva
- Integracija Transakcionih Verifikacionih Modula (TVM) od raznih ponuđača
 - plug and play
- Mnogi EDA (Electronic design automation) ponuđači ga podržavaju

Zašto TLM

- Dizajniranje elektroniskih sistema uključuje
 - Apstraktne ideje
 - Implementacija apstrakcija pomoću konkretnih gradivnih elemenata (log. kapije)
 - Sve do nivoa silicijuma
- RTL modeling je bazirano na diskretnom vremenu
 - Koristi čvorove, kašnjenja, događaje, ivice takta...
- TLM može bit modelovan u vremenu ili van njega
 - Ne mora se spuštati na bitski nivo
 - Procesi komuniciraju preko transakcija i poziva funkcija
 - Ne mora se ići do nivoa pina
- Simulaciono vreme
 - Gate Level : dugo traje
 - RTL Level : srednje traje
- TLM level: brzo (ubrzane funkcionalne simulacije)
 - Veće mogućnosti funkcionalne simulacije

Evolucija ASIC dizajna

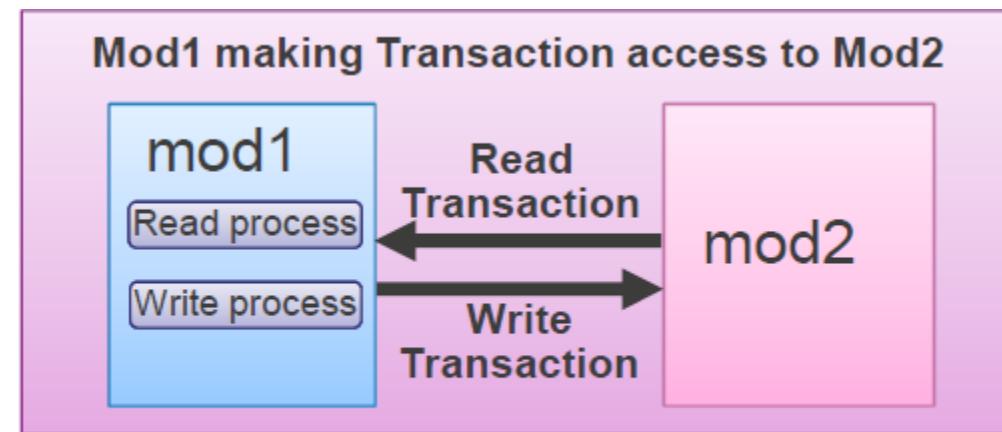
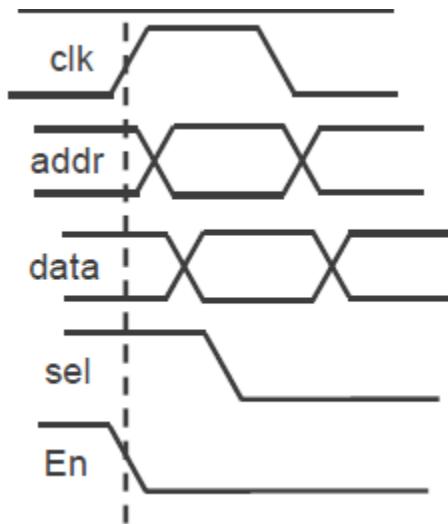




Šta je transakcija

- Transakcija je bilo koji transfer podataka između blokova
- Transfer podataka može da podrazumeva nekoliko ciklusa i signalnih tranzicija
 - primer: AHB Bus Read/Write
- Konvencionalni transferi na pin nivou
 - prednosti : vremenski tačno na nivou ciklusa takta (Cycle accurate), tačno modelovanje događaja na pin nivou, blisko realnosti
 - nedostatci : sporo se simulira, troši puno CPU vremena, teže se debuguje
- Rad sa podacima na TLM nivou podiže nivo apstrakcije
 - prednosti : Transferi se lakše kreiraju, debuguju, rukuju
 - mane : Može da izostane vremenska tačnost (Cycle Accurate?), distancirano od modelovanja na pin nivou, nije sintetizibilno

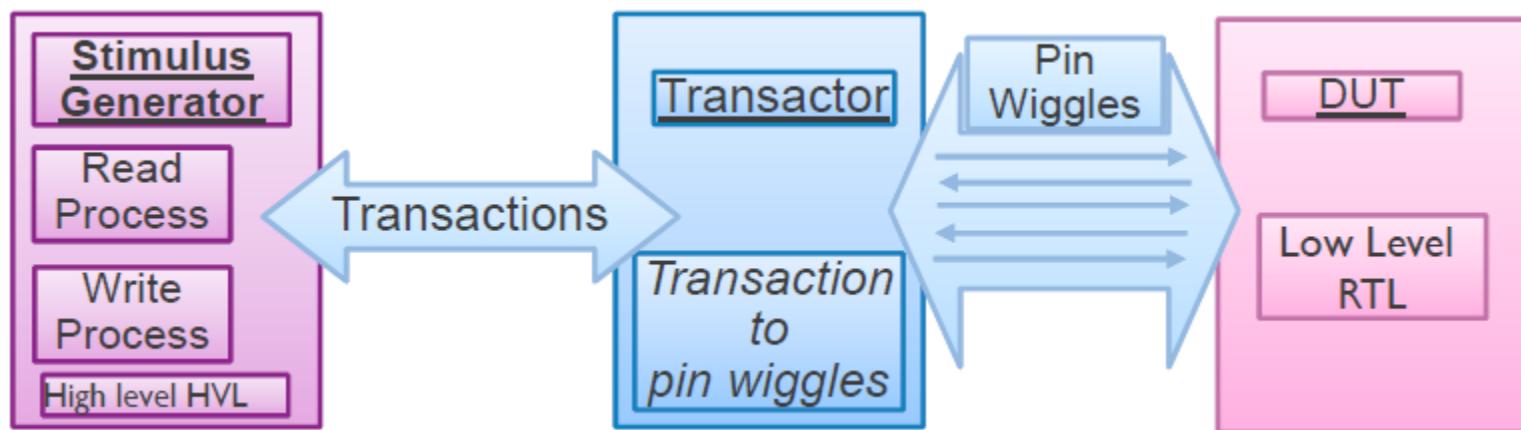
Konvencionalni nasuprot TLM



Verifikacija bazirana na transakciji (TLM based verification)

- Transaktori generišu transakcije
- Transaktor translira transakcije dole na pin nivo i nazad
- Stimulus generator aktivira read/write proces ka transaktoru
- Transaktor interpretira read/write i translira ih na pin nivo
- DUT ili Stimulus generator imaju zajednički reusable intefejs

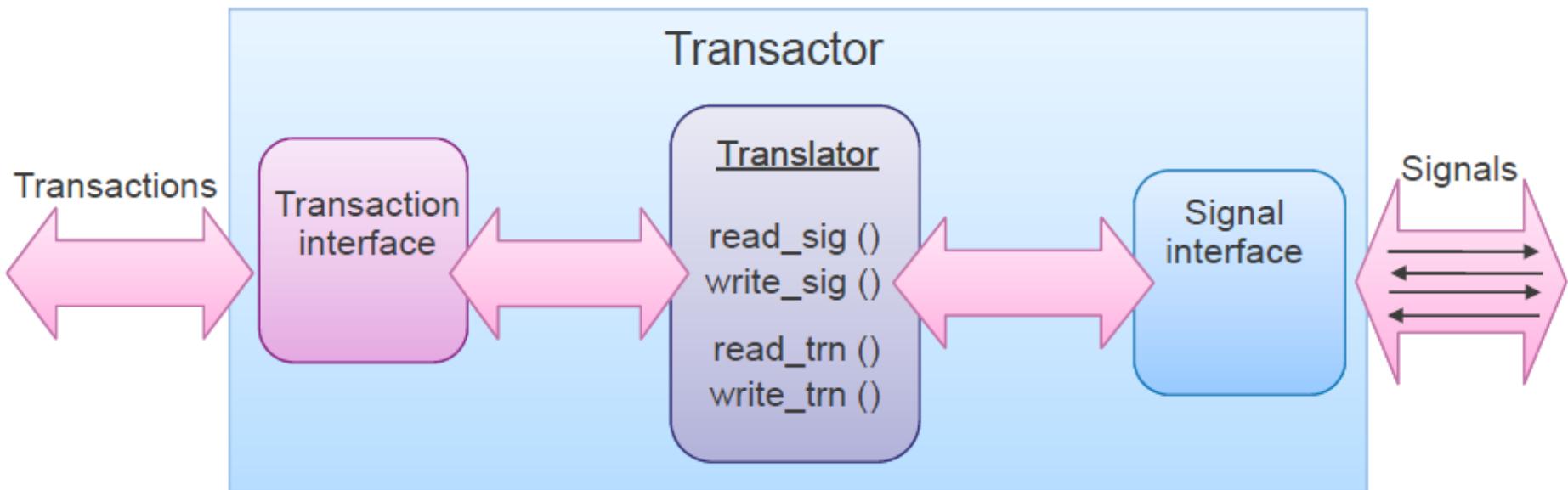
TLM i pin nivo



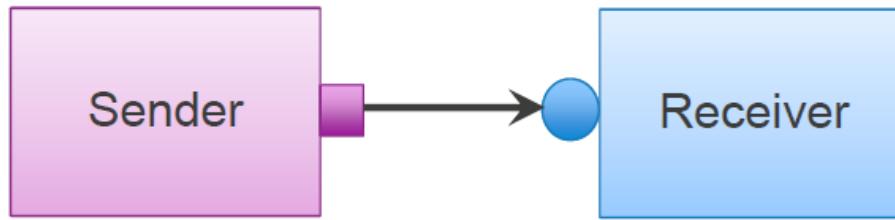
Transaktor

- Transaktor ima tri dela
- Transakcioni interfejs
 - Povezan na transakcioni generator/prijemnik
- Signalni interfejs
 - Povezan na signalni/pin-transfer generator/prijemnik
- Transaktor (Translator)
 - Transaktor ima funkciju “Translatora”
 - Kreira sekvence podataka za gonjenje signalnog interfejsa
 - Proverava transakcione/signalne intefejse po pitanju ispravnosti transfera

transaktor



TLM ilustracija



```
class sender;
    write_interface write_port;
task run ();
    write_port.write(24);
endtask
endclass : sender
```

```
virtual class write_interface;
    pure virtual task write (int val);
endclass
```

```
module top;
    sender se;
    receiver re;
    initial begin
        se = new();
        re = new();
        se.write_port = re ;
        se.run();
    end
endmodule : top
```

```
class receiver extends write_interface;
    task write (int val);
        $display ("Receiving %2d ",val);
    endtask : write
endclass : receiver
```

Receiving 24

OCP Term	Timing Accuracy	Abstractions	OSCI-TLM-2.0 equivalent
TL0	Cycle accurate	None, this is the RTL level	SystemC synthesise-able subset
TL1	Can be fully cycle-accurate, requiring clock synchronisation between bus master and bus slave, and respecting the OCP protocol. All beats of a burst are modelled.	Wires and signals are not modelled	none so far
TL2	User selectable number of timing points per bus burst	No clock synchronisation therefore some non-determinism. Optional averaging of bus occupancy over bursts or parts of bursts. Flow control not modelled explicitly.	none so far
TL3	4 timing points per bus burst, bus occupancy determined only by 'data receiver'	No modelling of independent write data phases, no ability to model intra-burst timing effects, no distinction between address order within a burst (eg wrapping and incrementing bursts are equivalent)	Approx-timed (AT) nb_transport()
TL4	Minimum necessary to run software on a virtual platform	"Pure functional" representation of memory-mapped bus. No flow control or ordering effects are modelled.	Loosely-timed (LT) b_transport() and nb_transport()

pregled

- UVM će detaljno pokriti TLM modelovanje
- TLM je pristup visokog nivoa u modelovanju digitalnih sistema
 - Komunikacioni detalji se razdvajaju od implementacionih detalja funkcionalnih jedinica
- Fokus je na digitalnoj komunikacionoj infrastrukturi
- Koristi postojeće OOP koncepte
- Obezbeđuje više nivoa apstrakcije
 - DUT može biti TL0
 - Verifikacioni Blokovi TL3
 - Komunikaciona infrastruktura može biti na nivou TL2
- Brze simulacije, jednostavno debagovanje
- TLM ne opredeljuje jedinstven nivo apstrakcije, već opredeljuje tehniku modelovanja
 - To je transakcioni nivo modelovanja, ne RT-nivo ili Behavior nivo