Sistem Verilog

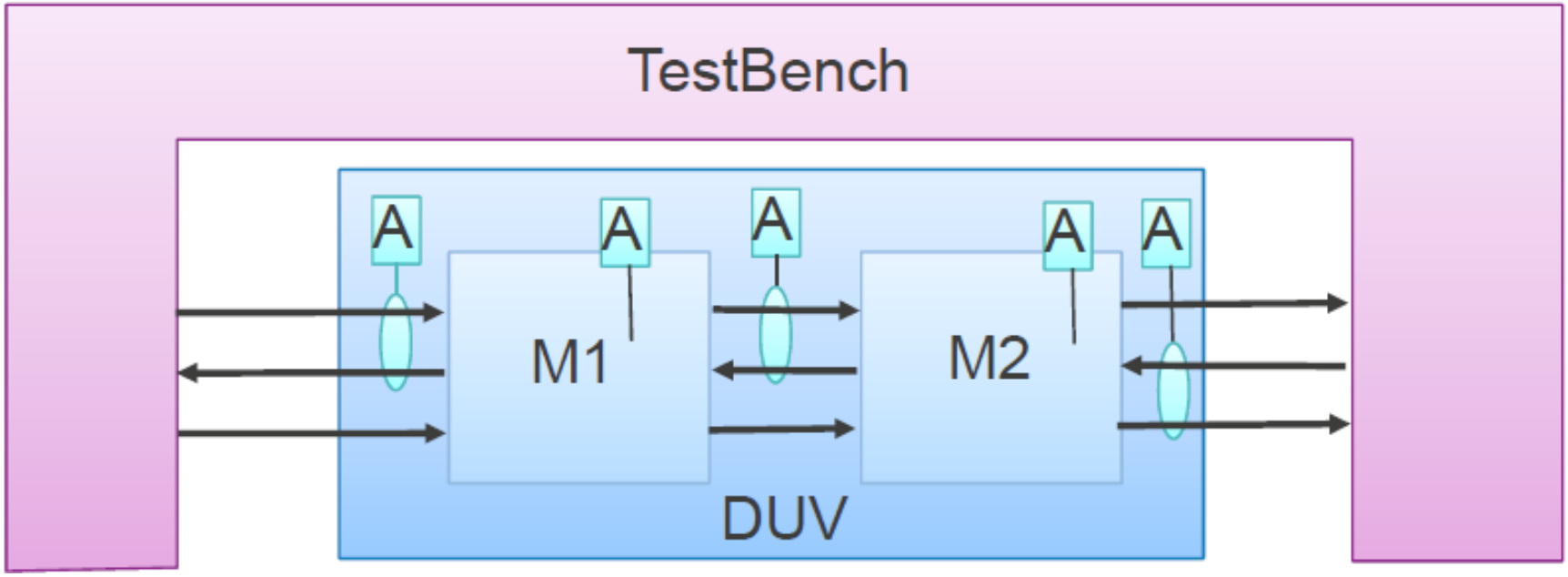
Assertions (tvrdnje)

- Cilj
  - Razumeti prednosti korišćenja Assertion-a u funkcionalnoj verifikaciji
- Sadržaj
  - Šta i kako
  - Assertions
  - Sekvence
  - Property
  - Implikacioni operatori
  - Ponavljanje (Repetitions)
  - BINDING

Šta je assertion?

- Assertion predstavlja logičku tvrdnju postavljenu unutar DUT-a ili spolja u njegovom okruženju koja verifikuje DUT.
- Assertion je logička tvrdnja koja može biti tačna ili netačna (true\false) postavljena u program, pri čemu dizajner očekuje tačnost te tvrdnje
- To je kod za verifikaciju određene karakteristike
- Proverava specificirano ponašanje sistema
- Tipično se izvodi iz tehničke specifikacije dizajna

Postavljanje assertion-a

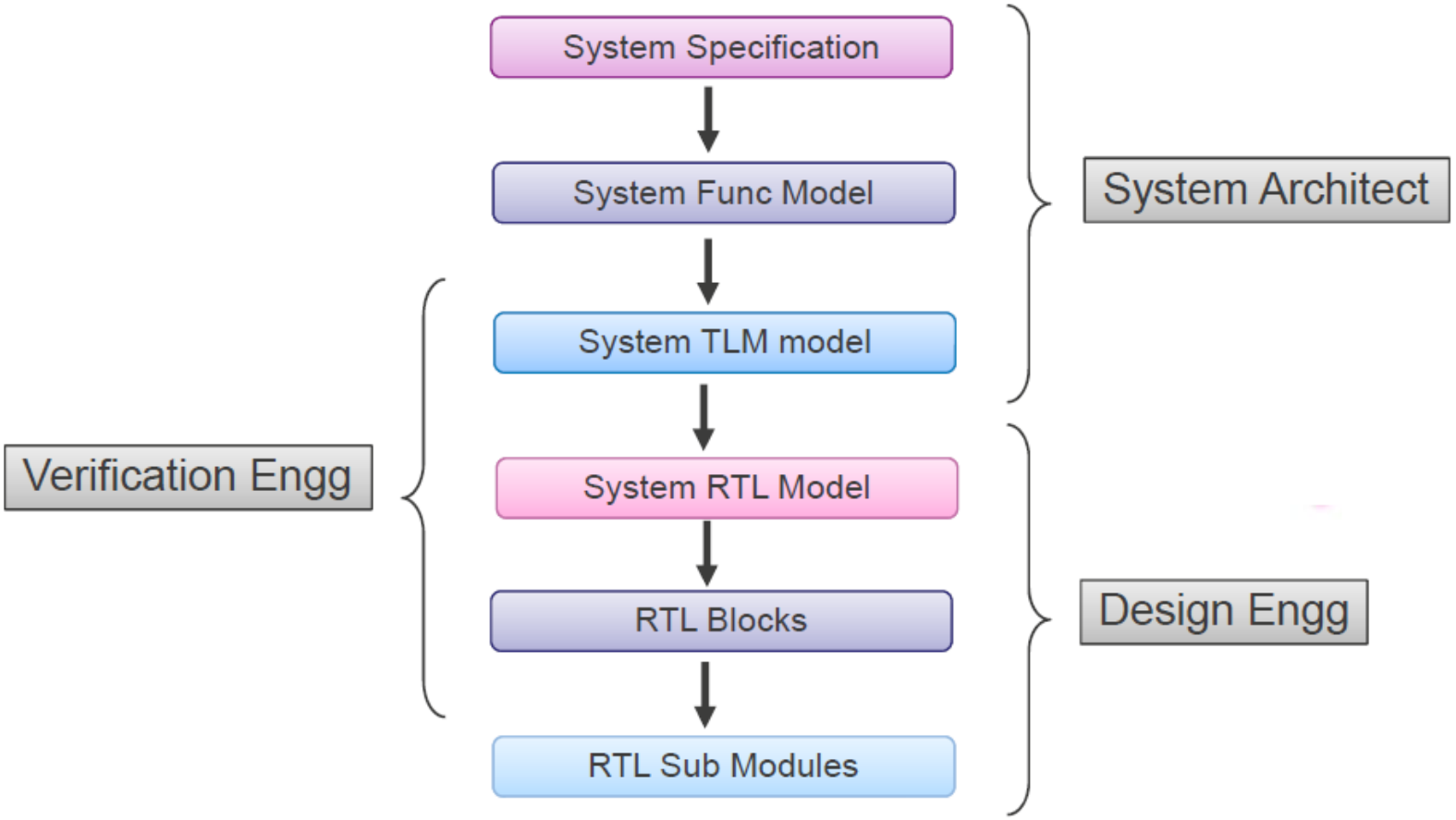


Zašto tvrdnje (assertions)

- Assertions se koriste da specificiraju uslove za koje programer podrazumeva da su tačni
- Prednosti upotrebe Assertion-a:
  - Povećavaju observabilnost.
  - Skraćuju vreme debagovanja.
  - Bagovi se ranije pronalaze i lakše se izoluju.
  - Kontrolišu nivo oštrine verifikacije.
  - Mogu da interreaguju sa C funkcijama.
  - Povezuju dokumentaciju i specifikaciju dizajna.

- Današnji ASIC dizajn
  - Procenat uspešnih ASIC tape-out-a je u značajnom opadanju
  - Rade se ponovni pokušaji (Re-spins)
- ASIC postaju sve više kompleksni
- Zašto su u opadanju?
  - Nisu dovoljno verifikovani
- Assertions su aktuelan trend u cilju bolje verifikacije kompleksnih čipova
- Osnovna ideje verifikacije se svodi na verifikaciju osnovnog cilja
- Današnji kompleksni projekti imaju preko 20,000 linija assertion koda
- Naredne generacije će imati preko 100,000 linija assertion koda
- U tako velikoj količini assertiona
  - Teško je njima rukovati
  - Teško ih je verifikovati
- Alati za rad sa assertion-ima. (0-In Formal, InFact, OneSpin, …)

Ko treba da piše assertion-e



Verifikacioni naspram dizajn assertions

Verifikacioni inženjer	Dizajn inženjer

• Verifikacija visokog nivoa
  • SystemLevel, Unit level
	• Verifikacija niskog nivoa
	  • Module/Submodules

• ulazi u detaljan Test Plan	• ne ulazi u detaljan Test Plan
• lako se prati	• lako se prati
• skraćuje vreme debagovanja	• skraćuje vreme debagovanja

•Kroz simulacije je najlakše naučiti koristiti assertion-e

•Assertion-i mogu da uspore simulaciju, ali skraćuju vreme debagovanja

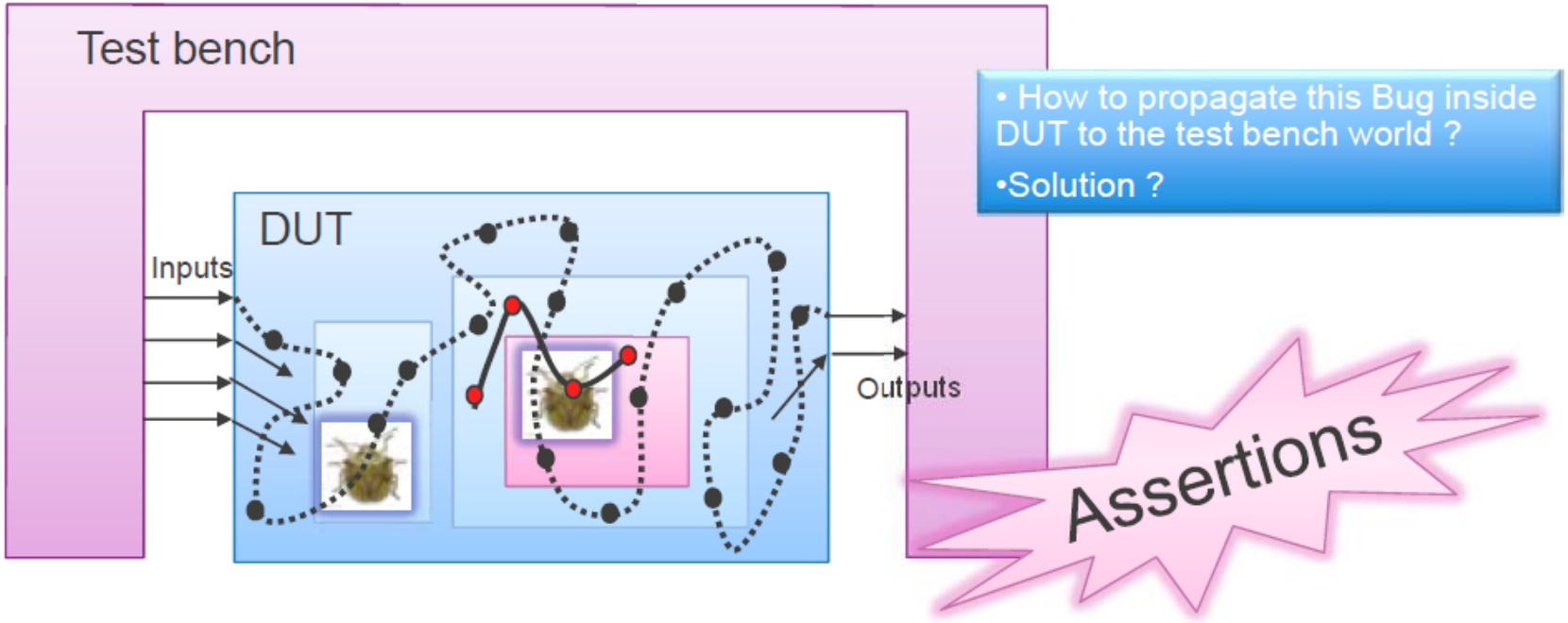
•Oni upućuju inženjere da više razmišljaju o bagovim

Aktuelne Assertion tehnologije

- Aktivno se koriste sledeći Assertions jezici
- OVL (Open Verification Library)
  - Besplatan
  - Koristi vhdl/verilog
  - Ne zahteva ništa dodatno preko vhdl/verilog simulatora
- PSL (Property Specification Language)
  - od 2005 ušao je u IEEE standard,
  - Postoje PSL/Vhdl i PSL/Verilog
- **SVA (SystemVerilog Assertions) – ovo je naša tema**
  - Deo su SV, prilagođeni su Verilog korisnicima!
  - Osnovni cilj je pokrivanje logičkih izraza
- Pažnja!
  - Treba biti veoma oprezan po pitanju šta se želi uraditi
  - Lažni uspeh / lažni neuspeh može da zavara inženjera – skupo košta

Konvencionalna verifikacija

- Pristup odozdo na gore
  - Prvo se verifikuju pod moduli, zatim viši hijerarhijski nivoi
- Top-level verifikacija podrazumeva
  - Pod moduli verifikovani, očišćeni od bagova.



Šta assertioni verifikuju

- Conditional : proveravaju ispravnost logičkih uslova korišćenjem Boolean izraza
- Sequence : proveravaju da li se sekvencijalno ponašanje odvija ispravno, korišćenjem temporal izraza
- Signal : proveravaju tipove signala
- X detection : mogu da detektuju da li je signal ne konektovan ili u koliziji
- Encoding types: proverava da li je enkoding ispoštovan (Onehot, gray code...)

Gde se assertion-i koriste

- Reaktivni testbenčevi mogu se napraviti sa akcionim blokovima
  - Može se aktivirati task zavisno od uspeha/neuspeha assertiona
- Mogu se deklarisati u SV unitima
- Mogu se povezati sa verilog/vhdl modulima
- Koriste se za dinamičku i formalnu verifikaciju
- Koriste multi/single klok domen sekvence
  - Power shut down/wake-up se može verifikovati sekvencama

Gde se assertioni ne koriste

- Race Conditions
- Provera ekvivalencije
- Pokrivensto koda

Gde se assertioni koriste

- HDL Assertioni (dizajnerski zahtevi)
  - unutar modula
  - FIFO, granični slučajevi, nedozvoljene tranzicije stanja..
- Module Interface Assertioni (dizajn interfejsi)
  - Između modula (verifikacija protokola)
  - Slave Master interfejsi, Bus protokoli, verifikacija potvrda..
- Chip Interface Assertioni
  - Tokom integracije proizvođačkih IP-jeva i njihove integracije
- Assertioni u proveri performansi
  - Kašnjenja, broj ciklusa čekanja za potvrdu, burst, pisanje/čitanje…

Dve vrste assertiona u SVA

- Trenutni (Immediate) Assertion
- Konkurentni Assertion

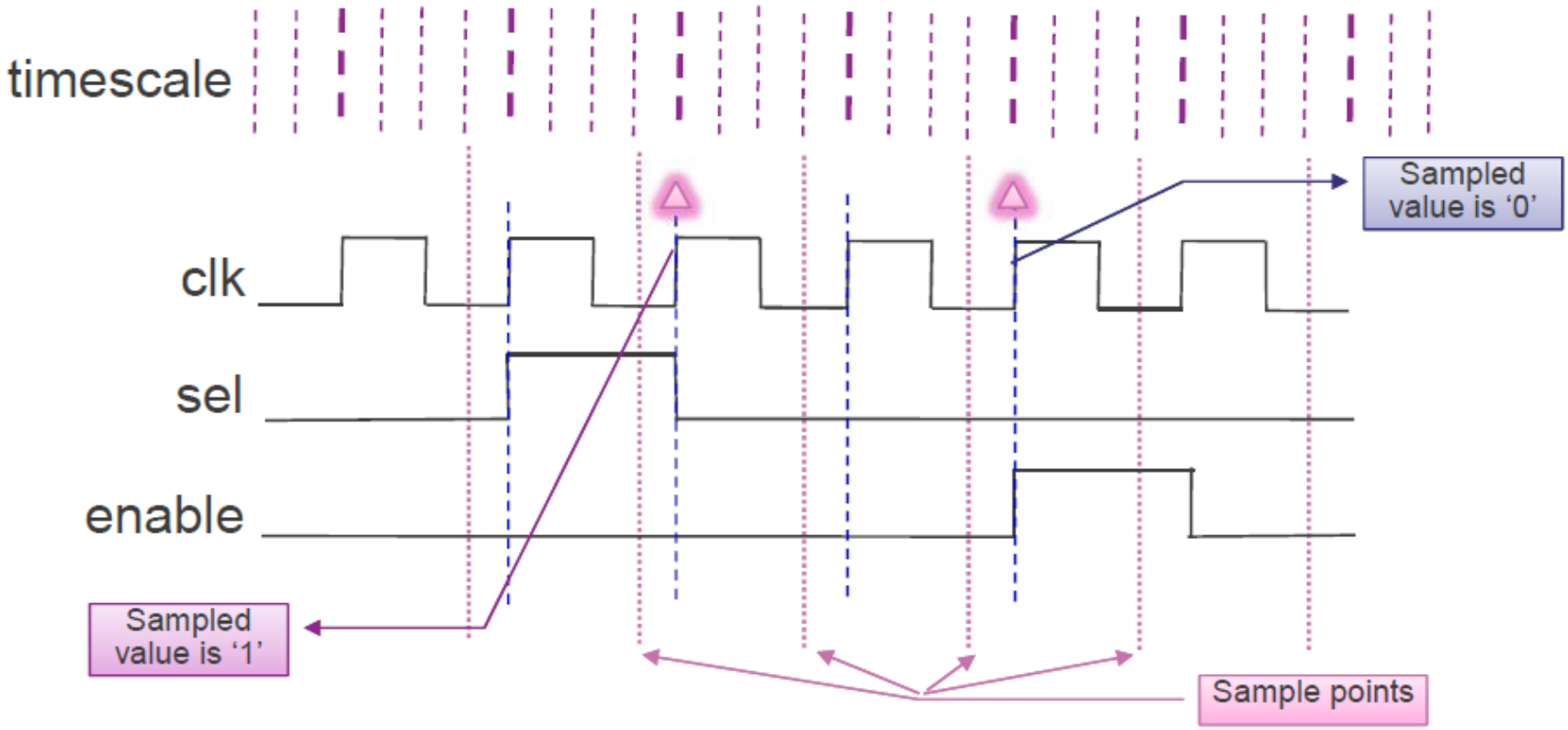
Immediate Assertions	Concurrent Assertions
Based on simulation event	Based on clock cycles
Used only with dynamic simulation	Used with formal and dynamic simulation
No use of property keyword	property keyword used
Have to be placed in procedural block	Placed in procedural block, modules, interfaces and program definitions
Evaluated immediately	Evaluated on clock edges

```
always_comb begin
  a_name: assert (a && b); end
```

```
a_name : assert property (@
      (posedge clk) not (a && b));
```

Semplovanje u SVA

- Kada se signali sempluju u SVA ?
- Signali se sempluju na početku svakog vremenskog koraka (vrednost vremenske rezolucije)
  - U SV vremenska rezolucija se definiše: `timescale 1ns/1ps



Trenutni (Immediate) assertions

- Izvršavaju se kao izrazi u proceduralnom bloku
- Interpretirani su kao na primer izrazi u okviru proceduralnog if bloka
- Ako se izraz sračuna i dobije vrednost
  - 1, interpretira se kao tačan/assertion-uspešan
  - 0, X ili Z, tada se interpretira kao netačan/assertion-neuspešan

```
example_assert_immediate :
                    assert (b || c) $display ( "%m pass"); else $info(" %m fail");
```

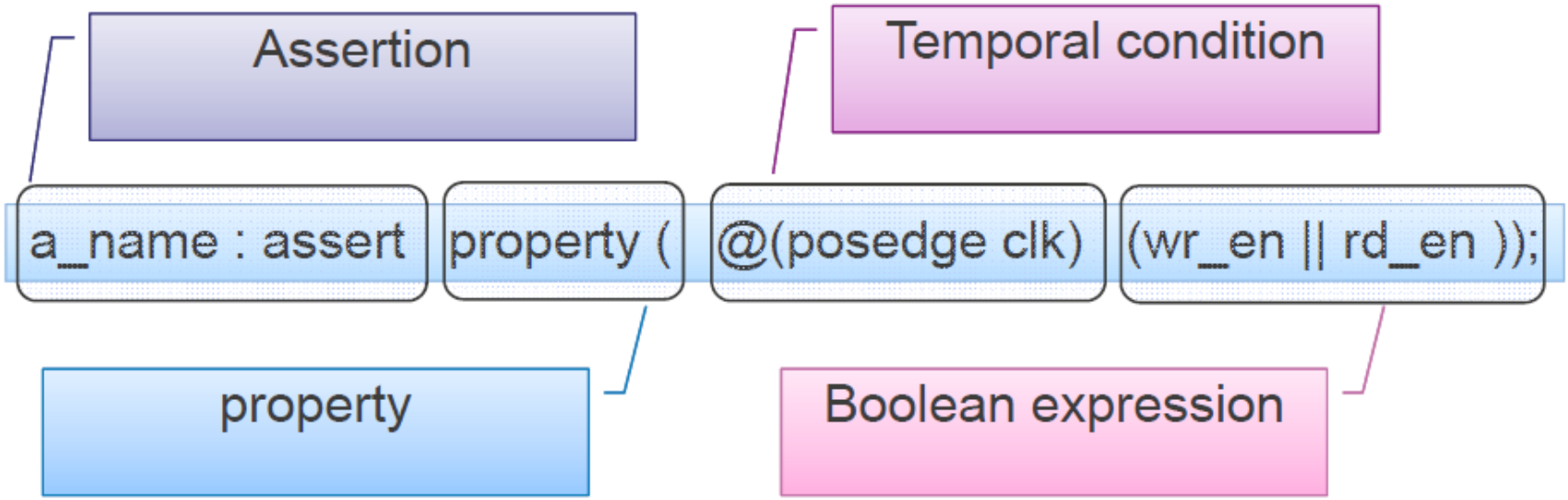
```
# ass13.example_assert_immediate      pass
# ** Info: ass13.example_assert_immediate      fail
#    Time: 3630 ns  Scope: ass13.example_assert_immediate File: assertion13.sv Line: 22
```

Severity Levels:

$fatal	→ Run-time fatal error; simulation stops
$error	→ Run-time error; simulation stops
$warning	→ Run-time warning; simulation doesn't stop; tool specific too
$info	→ Assertion failure carries no specific severity !

Konkurentni assertion

- Opisuje ponašanje postavljeno u vremenu
- Signali se sempluju u okviru preponed regiona SV raspoređivača, dok se assertion vrednost proračunava u okvireu observe regiona



Sekvence

- Boolean logika u osnvnom obliku ne pokriva koncept vremena
- Temporal logika : Boolean logika prezentovana u vremenu
- Sekvence dozvoljavaju da se definišu temporal izrazi
- Sekvence omogućavaju formiranje i manipulaciju sekvencijalno ponašanje
- linearne sekvence:
  - Lista SV Boolean izraza koji se izvršavaju u linearnom redosledu prolaska vremen
  - Ako je 1. izraz tačan pri prvom taktu, zatim 2. izraz tačan pri sledećem taktu, zatim 3…
- Sekvence se mogu slagati konkatenacijom , slično kao konkatenacija listi
- Može da se specificira kašnjenje od kraja prve do početka druge sekvence
- ##5 formira kašnjenje u broju ciklusa – konkretno 5 ciklusa (Ne treba mešati sa #5 što predstavlja čekanje od 5 vremenskih jedinica)

Definisanje vremenskih jedinica podsetnik

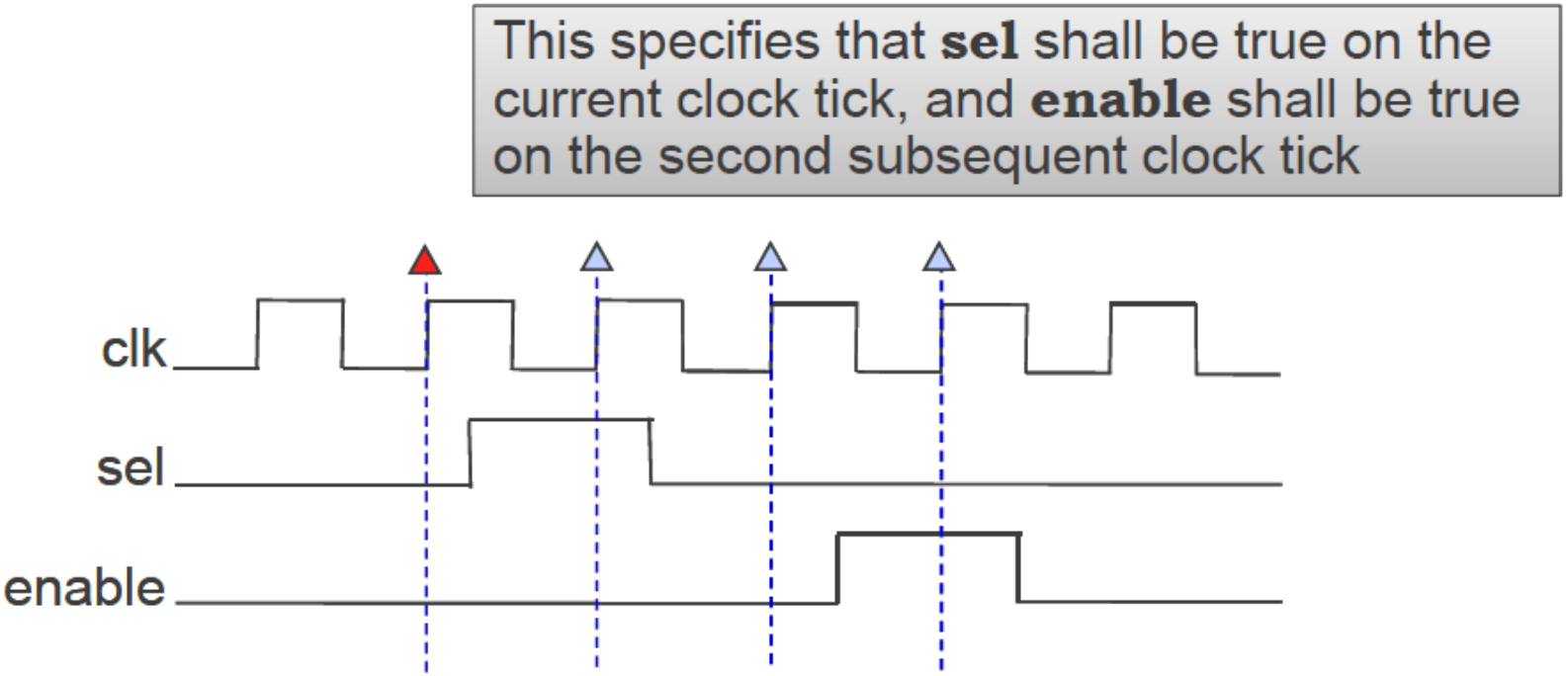
- `timescale 100ps/10ps
  - Definiše #1 kašnjenje da bude upravo 100ps
  - Dok se rezolucija svodi na desetinu, jer je 100ps/10ps=10, dakle minimalno kašnjenje može se postaviti kao #0.1, što će biti jednako 10ps. Sitnije rezolucije će se zaokruživati na ovde definisano.
- `timescale 1ns/1ps
  - Definiše #1 kao 1ns, dok se rezolucija svodi na #0.001 jer je 1ns/1ps=1000.

Sekvence

- Gde es deklarišu sekvence?
  - U Modulima
  - Interfejsima
  - Programima
  - Clocking blokovima
  - Package-ima
- Sekvence se mogu referencirati u
  - Property-ju, assertionu ili cover izrazu

Sekvence sa fiksnim kašnjenjem

- ## praćeno brojem, specificira kašnjenje nakon date ivice takta
- primer: sel ##2 enable



Specificiranje kašnjenja

- Kašnjenje ## u ciklusima takta
- Fiksno
  - ##1
  - ##0 (spaja dve sekvence bez kašnjenja),
- Može biti interval
  - ##[1:3] , ##[0:5], …
- Otvoreni interval
  - Primer: ##[1:$], ##[5:$]
  - Od 1 ciklusa do kraja simulacije, ili od 5-og cikljusa do kraja simulacije
  - $ označava beskonačnost

Sekvence i podsekvence

- Sekvence se mogu referencirati unutar druge sekvence
- Ciklična zavisnost nije dozvoljena!

```
sequence s1;
    a ##1 b ##1 c ;
endsequence
sequence s2;
    @(posedge clk) d ##1 s1 ##1 e;
endsequence
```

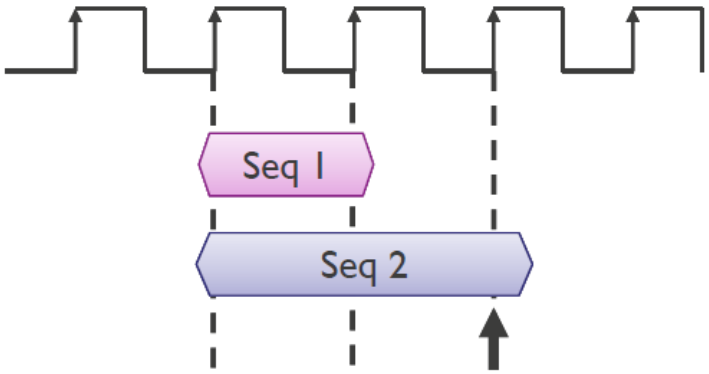
Cyclic dependency

```
sequence s3;
    @(posedge clk) a ##1 s4 ;
endsequence
sequence s4;
    @(posedge clk) b ##1 s3 ;
endsequence
```

```
sequence s5;
    @(posedge clk) d ##1 a ##1 b ##1 c ##1 e;
endsequence
```

Logički uslovi sa sekvencama (I)

- Obe sekvence startuju istovremeno
- Poređenje se realizuje kada se poslednja sekvenca završi

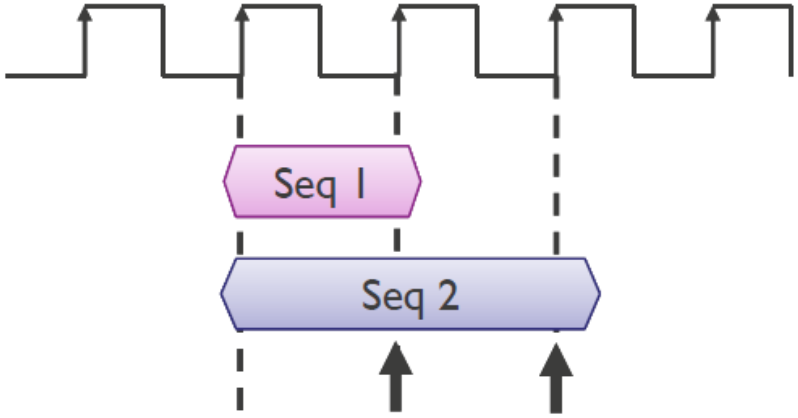


Seq1 **and** Seq2

primer: (a ##1 b) and (c ##1 d ##1 e)

Logički uslovi sa sekvencama (ILI)

- Obe sekvence startuju istovremeno
- Poređenje se realizuje kada se poslednja sekvenca završi

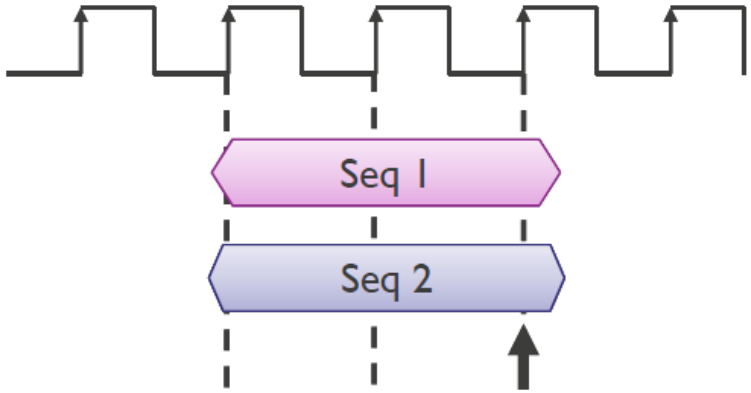


Seq1 **or** Seq2

primer: (a ##1 b) or (c ##1 d ##1 e)

Preklapanje sekvenci

- Obe sekvence startuju istovremeno
  - Završavaju se istovremeno
- Poređenje se realizuje na kraju

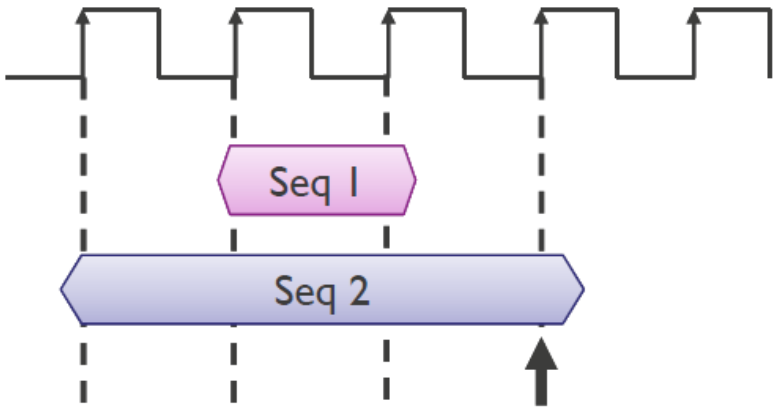


Seq1 **intersect** Seq2

primer: (a ##1 b) intersesct (c ##1 d)

Sekvenca unutar sekvence

- Sekvenca u potpunosti obuhvata drugu sekvencu

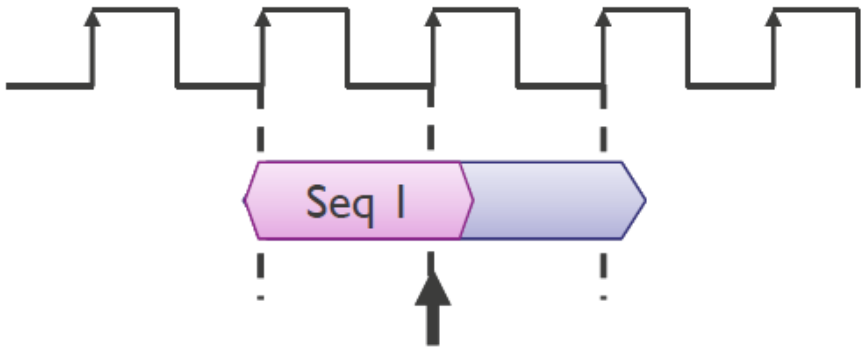


Seq1 **within** Seq2

primer: (a ##1 b) intersesct (c ##1 d ##1 e ##1 f)

Prvo pojavljivanje first_match Seq1

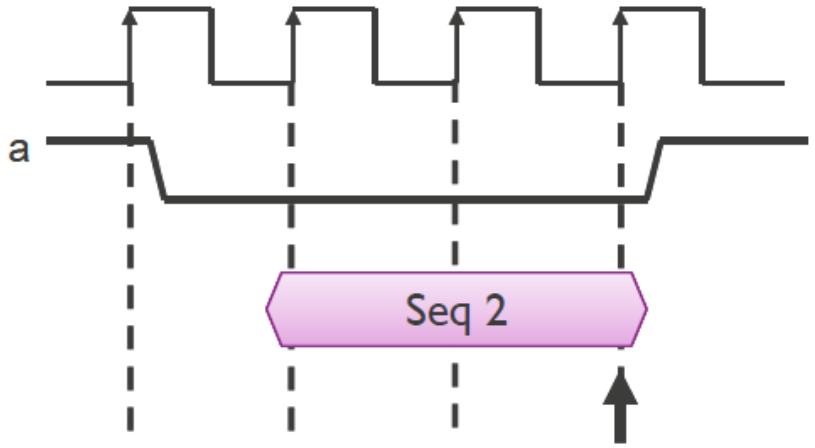
- Obe sekvence startuju istovremeno



first_match Seq1

Duž sekvence / sequence throughout

- Throughout operator zahteva da izraz bude istinit duž cele sekvence



!a **throughout** Seq2

•Assertion based functions
•These functions are not limited to assertions
•Can be used in procedural code too

Disable iff

- disable iff izraz se može koristiti uz property
- disable iff se još naziva reset izraz
- Dozvoljava specificiranje sihronog reseta
- Ugnježdivanje disable iff nije dozvoljeno

```
ex_disable_iff : assert property (disable iff (abort) seq1 );
```

```
@(posedge clk) disable iff (reset) (a |-> !b);
```

Sistemske funkcije

- Funkcije za korišćenje u assertionima, ali i u proceduralnom kodu

Functions	Description
$sampled	Returns sampled value of expr
$rose	A transition from low to high, returns true
$fell	A transition from high to low, returns true
$stable	Returns true, if the value of expr did not change
$past	Past values (with number of clock ticks)
$onehot	Returns true if only one bit of expr is high
$onehot0	Returns true if at most one bit of expr is high
$countones	Count number of 1s in a bit vector expr
$isunknown	Returns true if bit of expr is X or Z

Sekvence sa dva izvora takta

- Sekvence sa dva izvora takta su dozvoljene

```
@(posedge clk_1) a1 ##2 @(negedge clk_2) a2;
```

```
@(posedge clk_1) a1 ##0 @(posedge clk_2) a3;
```

```
property mult_clk;
      @(posedge clk_1) a1 ##1
                @(posedge clk_2) a2 ##1
                          @(posedge clk_3)  a3;
endproperty
```

Assume - podrazumevati

- assume izraz dozvoljava properti-ju da budu podrazumevan i za formalnu analizu i simulaciju
- Kada je property podrazumevan (assumed)
  - Alati ograničavaju okruženje da drže property
- Kao bilo koji drugi assert property,
  - Podrazumevani property mora biti proveren i raportiran ukoliko se ne drži
- Alate ne mora da raportira uspešnost podrazumevanog property-ja

```
a21: assume property ( @(posedge clk) req inside {0, 1} ) ;
```

Cover property

- Cover iskaz se može koristiti za praćenje (monitoring) sekvenci/propertija
- Cover property formira jedan cover point.
- Alat prikuplja informacije
  - Sračunava i raportira rezultate na kraju simulacije
- Rezultati pokrivenosti su podeljeni u dva dela
  - Coverage za properties
  - Coverage za sekvence

```
sequence s1;
    @(posedge clk) a ##1 b;
endsequence
cover s1;
```

Coverage for Sequence:
- Number of times attempted
- Number of times matched

```
sequence  seq1 (ack);
    @(posedge clk) $rose(rdy) ##2 !ack;
endsequence
cover property (seq1(ifeed));

always cover property (seq2);
```

Coverage for Property:
- Number of times attempted
- Number of times succeeded
- Number of times failed
- Number of success due to vacuity

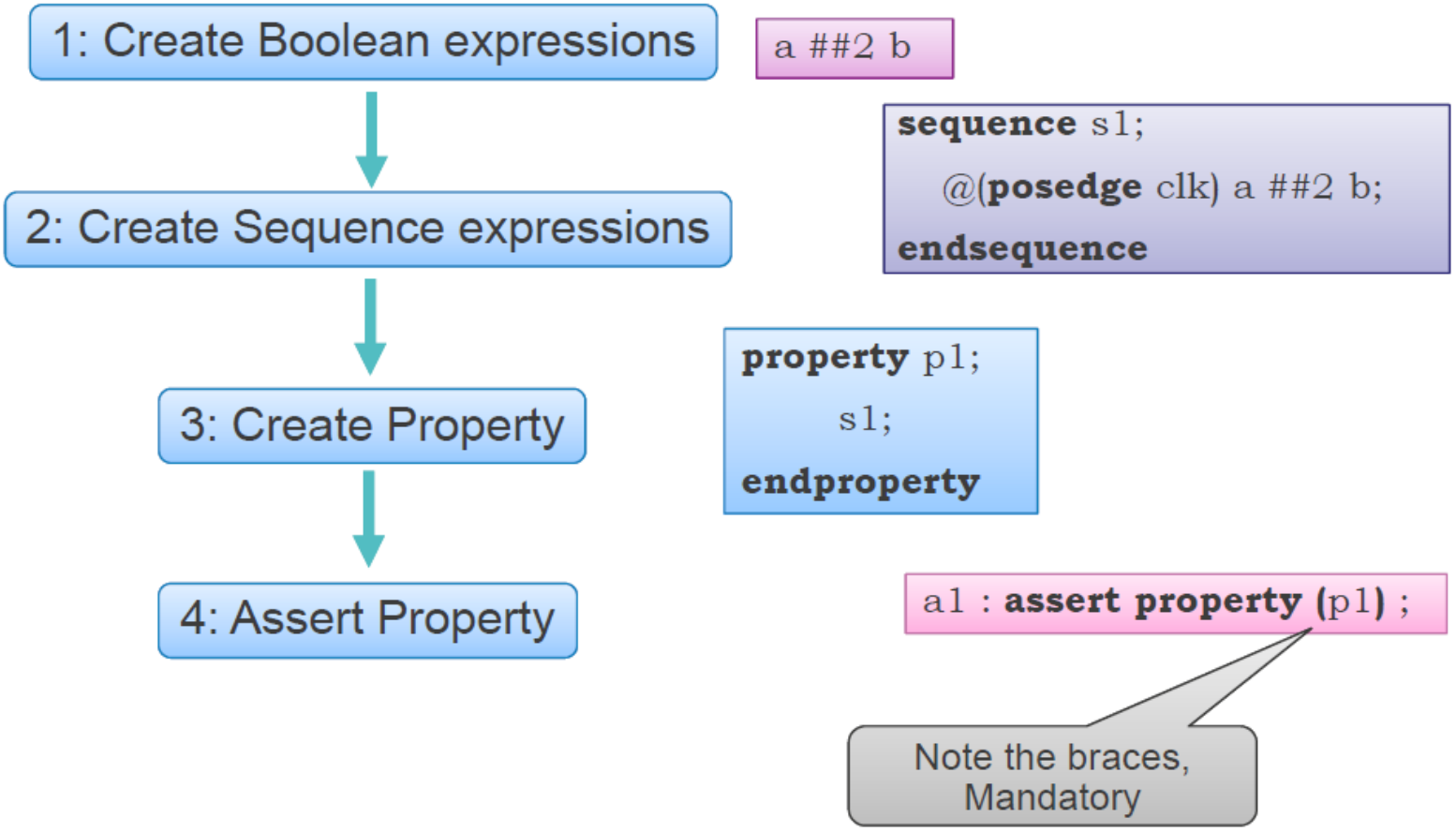
Expect iskaz

- expect iskaz startuje jedan thread za dati property
  - expect (property spec) else action_block ;
- Ako property padne (fails), opcioni else deo iskaza se izvršava
- Izvršava se po pozivu!

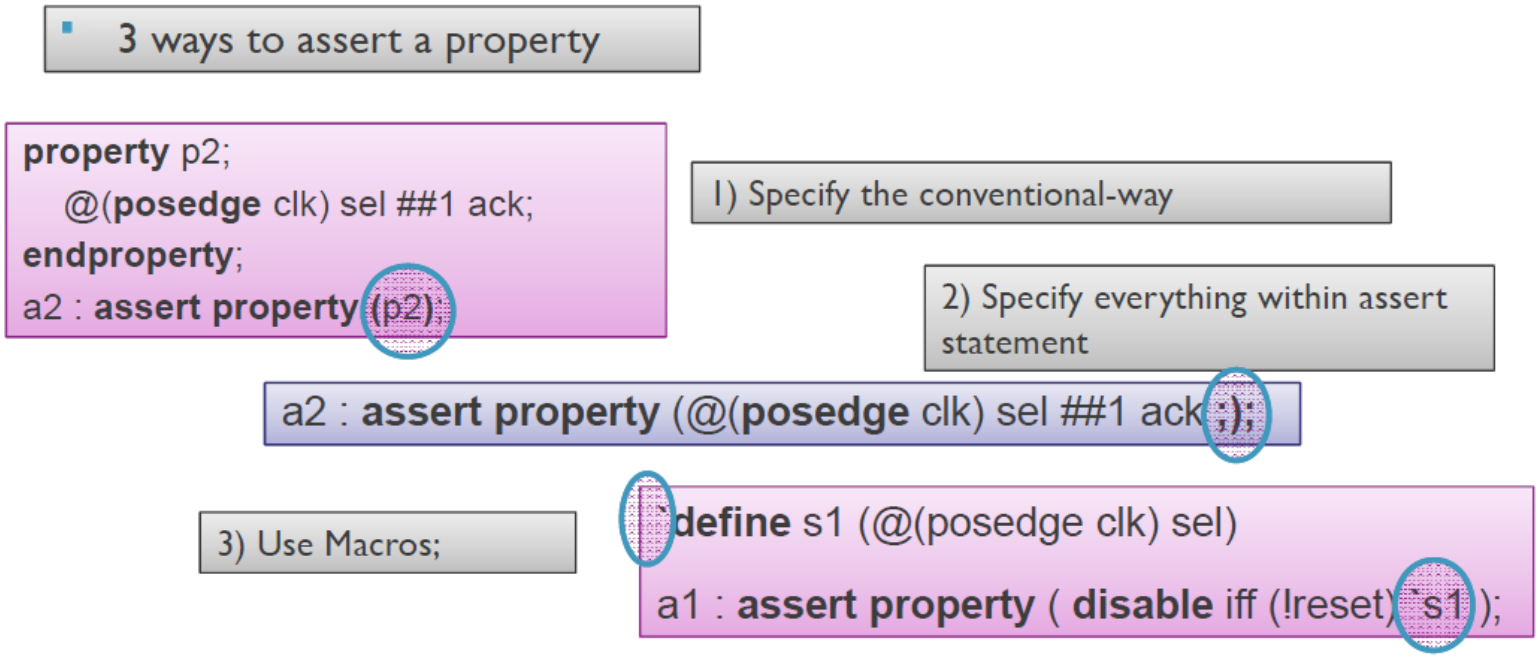
```
module tst;
   bit clk,a,b,c;
   initial for (int i=0; i <10; i++) #1 clk = ~clk;
   initial begin
      fork
      #1 a = 1; #2 a = 0;
      #2 b = 1; #3 b = 0;
      #3 c = 1; #4 c = 0;
      join
   end
   initial
      #1 expect( @clk a ##1 b ##1 c ) else $error( "expect failed" );
      // #2 … … … ERROR statement executed!!!
endmodule
```

```
# ** Error: expect failed
#    Time: 3 ns Started: 3 ns  Scope: tst File: expect.sv Line: 7
```

SVA osnovni gradivni elementi



Assertovanje property-ja



- Macro olakšava posao
  - Jedan property ili sekvenfa može se koristiti više puta, re-usable
  - Argumenti se mogu prosleđivati , korišćenjem arg
- Treba biti oprezan sa sintaksom.

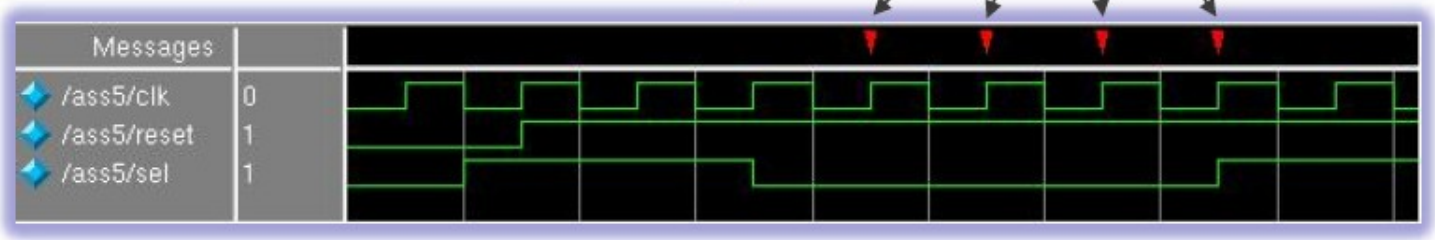
Primer sekvence

```
sequence s1;
    @(posedge clk) sel ;
endsequence

property p1;
     s1;
endproperty

a1 : assert property ( disable iff (!reset) p1);
```

Assertion Errors shown in the wave window

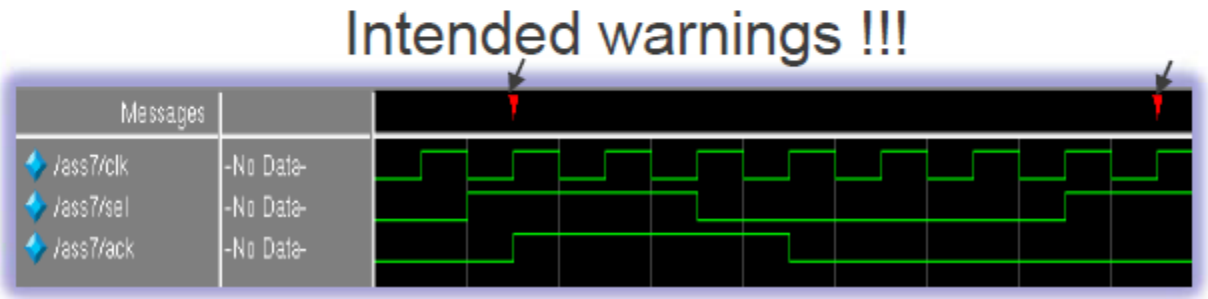



```
# ** Error: Assertion error.
#    Time: 9 ns Started: 9 ns  Scope: ass5.a1 File: assertion5.sv Line: 28
# ** Error: Assertion error.
#    Time: 11 ns Started: 11 ns  Scope: ass5.a1 File: assertion5.sv Line: 28
# ** Error: Assertion error.
#    Time: 13 ns Started: 13 ns  Scope: ass5.a1 File: assertion5.sv Line: 28
# ** Error: Assertion error.
#    Time: 15 ns Started: 15 ns  Scope: ass5.a1 File: assertion5.sv Line: 28
```

Operator implikacije “|->”

```
property p2;
   @(posedge clk) sel ##1 ack;
endproperty;
a2 : assert property (p2);
```


```
property p2;
   @(posedge clk) sel |-> ack;
endproperty;
a2 : assert property (p2);
```



- Property traga za validnim početkom sekvence na svaku rastuću ivicu takta clk
- Gornji primer, ako sel nije visok na rastuću ivicu clk, False warning se ispisuje!
  - Greška će se pojaviti, ako alat nije detektovao validan početak – rastuću ivicku clk
- Nasuprot tome, ako je cilj proveriti dok je sel visok, da li je ack visok !
- SVA implikacioni operator realizuje taj cilj
  - prethodnik |-> posledica
- Implikaciona konstrukcija unutar definicije property-ja, NE u sekvenci

Implikacija u istom ciklusu

- |-> (preklopljena implikacija -Overlapped Implication)
- Ako je uzrok ispunjen, posledica se sračunava u istom ciklusu
- Ako uzrok nije ispunjen implikacija je po definiciji tačna, bez obzira na istinitost posledice
- *Pravi uspeh*, ako je uzrok tačan i posledica tačna.
- Neuspeh, ako je uzrok tačan, a posledica netačna.

```
property p2;
   @(posedge clk) a |-> b;
endproperty;
a2 : assert property (p2);
```

If a is high on a posedge of clk, **then** b should also be high on the **same** clk edge

Implikacija u sledećem ciklusu

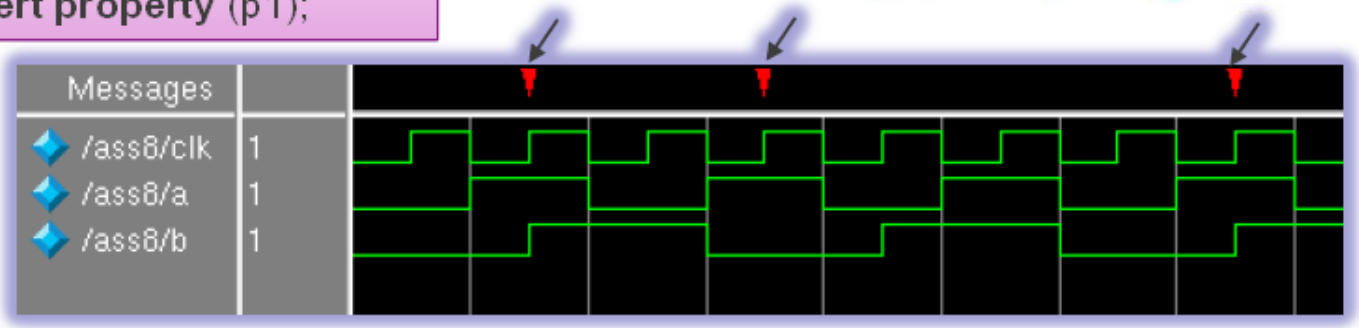
- |=> (Neprekloplјena implikacija - Non-Overlapped Implication)
- Ako je uzrok tačan, posledica se sračunava u sledećem ciklusu.
- Ako uzrok nije ispunjen implikacija je po definiciji tačna, bez obzira na istinitost posledice
- *Pravi uspeh*, ako je uzrok tačan i posledica tačna.
- Neuspeh, ako je uzrok tačan, a posledica netačna.

```
property p2;
   @(posedge clk) a |=> b;
endproperty;
a2 : assert property (p2);
```

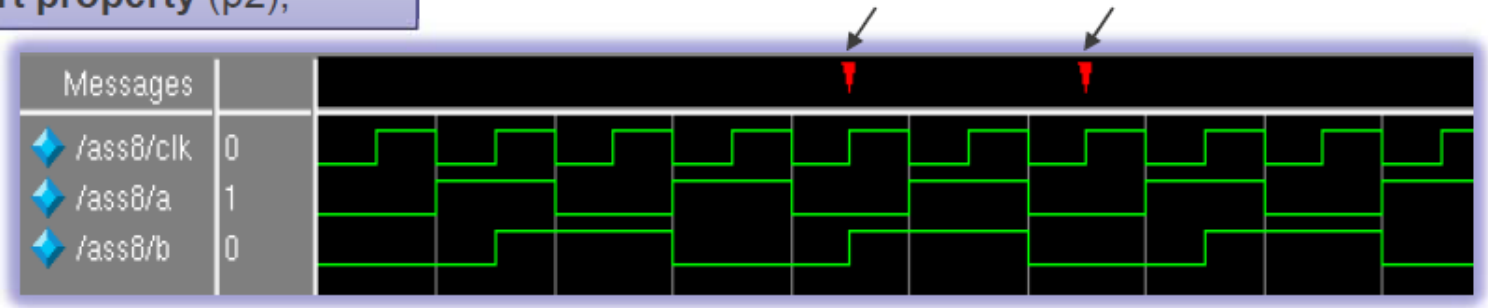
If a is high on a posedge of clk, **then** b should also be high on the **next posedge of clk**

Implikacije |-> nasuprot |=>

```
property p1;
   @(posedge clk) a |-> b ;
endproperty;
a1 : assert property (p1);
```

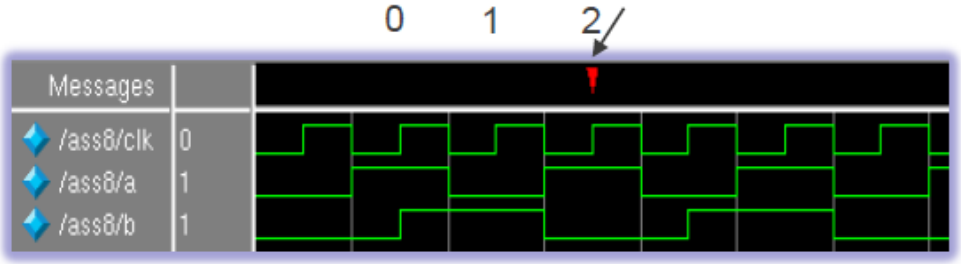



```
property p2;
   @(posedge clk) a |=> b ;
endproperty;
a2 : assert property (p2);
```

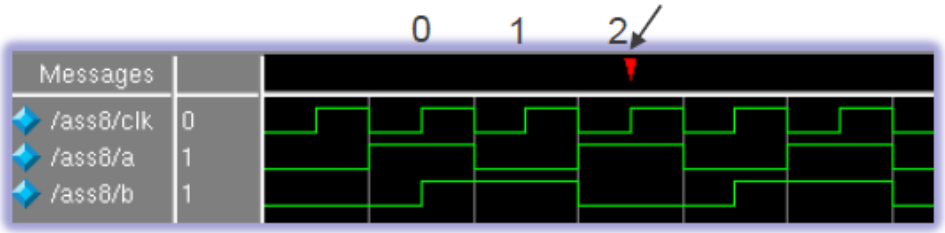


Crveni trougao markira netačnu implikaciju

```
property p3;
      @(posedge clk) a |-> ##2 b;
endproperty
a3 : assert property (p3);
```

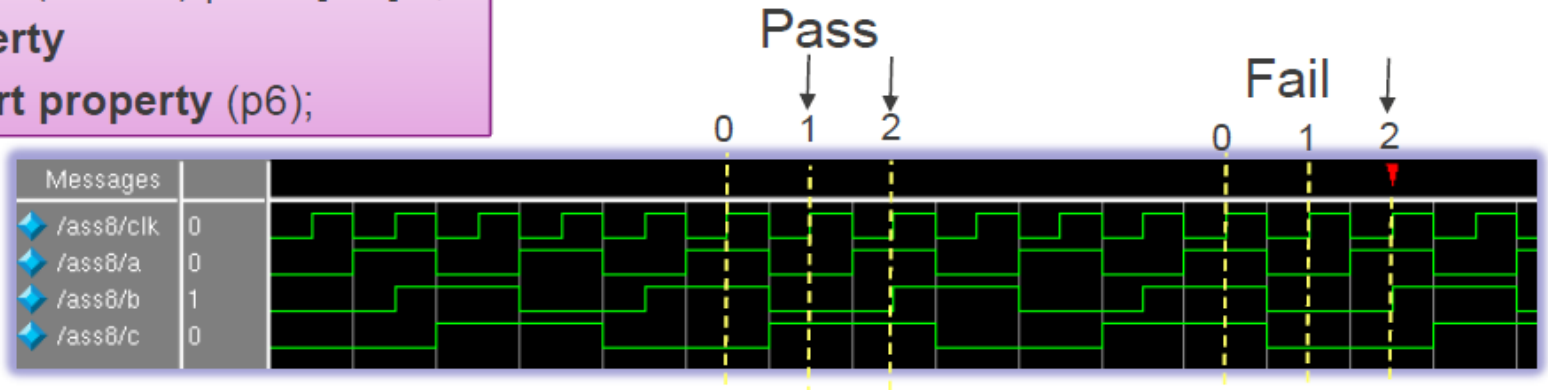



```
property p4;
      @(posedge clk) a |=> ##1 b;
endproperty
a4 : assert property (p4);
```




```
property p6;
      @(posedge clk)
            (a && b) |-> ##[1:2] c;
endproperty
a6 : assert property (p6);
```

Delay range : c can be high, either on the 1st clk edge or the 2nd clock edge.



Sekvence unutar iskaza - property-ja

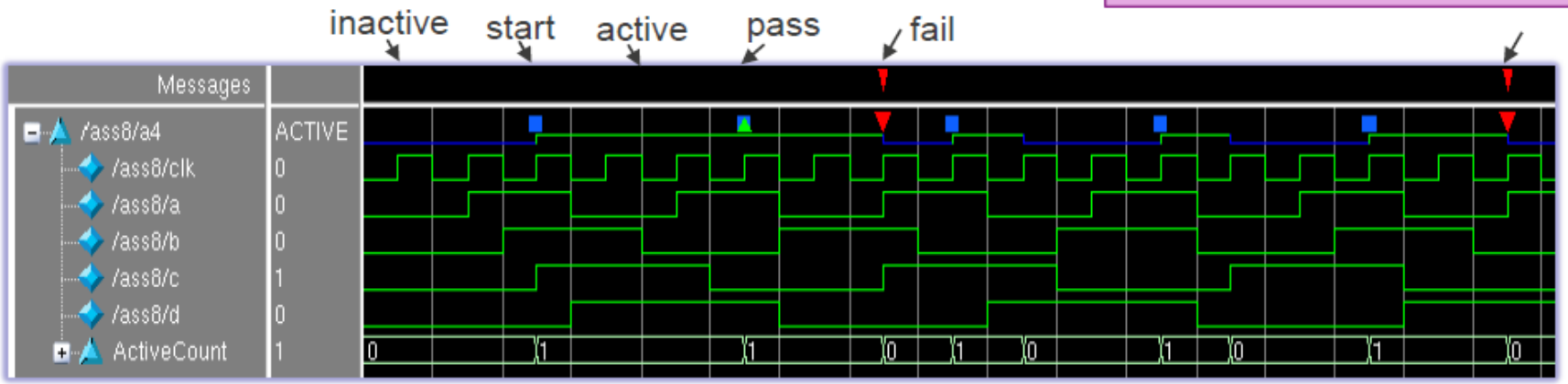
- Sekvenca može biti uzrok i posledica u okviru iskaza
- S1 na pozitivnu inicu takta clk, b prati a, nakon 1 ciklusa takta
- S2 na pozitivnu inicu takta clk, d prati c, nakon 1 ciklusa takta
- P4 sekvenca s1, implicira sekvencu s2 u sledećem ciklusu takta

```
sequence s1;
    @(posedge clk) a ##1 b;
endsequence
sequence s2;
    @(posedge clk) c ##1 d;
endsequence

property p4;
  @(posedge clk) s1 |=> s2;
endproperty
a4 : assert property (p4);
```

Simulator dependent (for the moment) !!!!

■ - Start ▲ - Pass ▼ - Fail

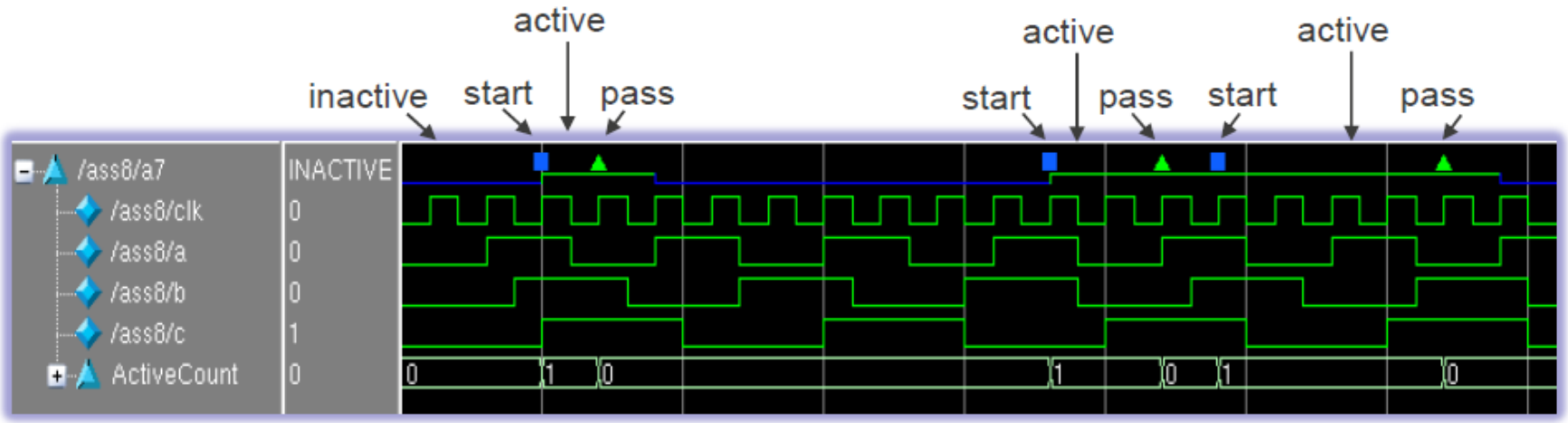


Operator $ u kontekstu intervala

```
property p7;
    @(posedge clk) (a && b) |-> ##[1:$] c;
endproperty
a7 : assert property (p7);
```

- Inactive, active, fail, pass
  - Means that we have concurrent assertions !

The checker will keep checking for a match until the end of simulation

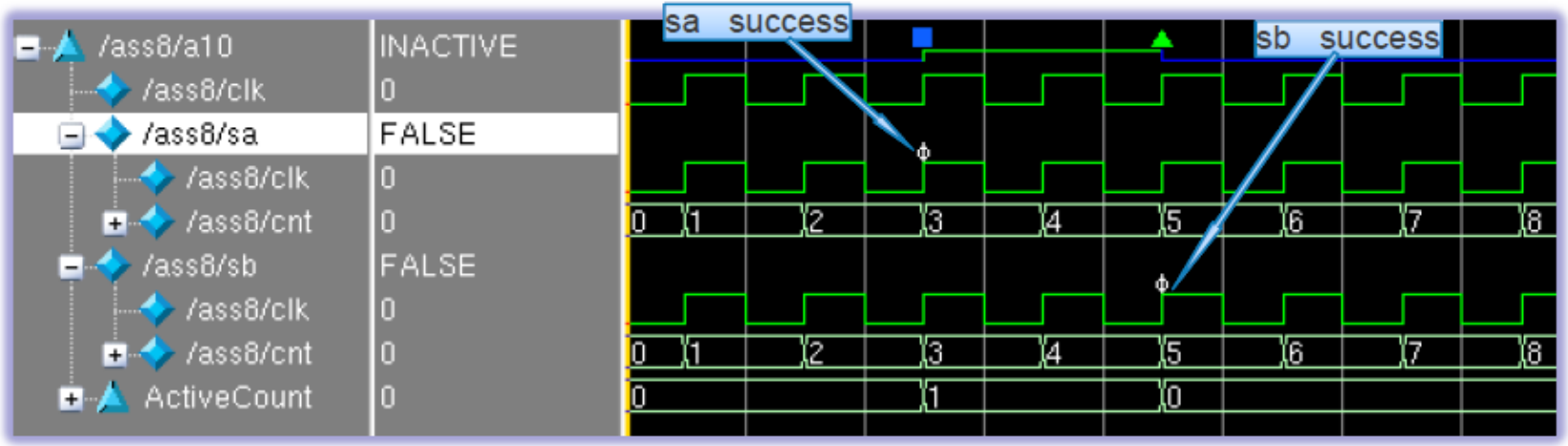


.ended konstrukcija

```
sequence sa;
    @(posedge clk) (cnt ==1) ##1 (cnt ==2);
endsequence
sequence sb;
    @(posedge clk) (cnt ==3) ##1 (cnt ==4);
endsequence
```

.ended → samples a seqence and returns boolean (TRUE/FALSE)

```
property p10;
    @(posedge clk) sa.ended |=> ##1 sb.ended;
endproperty
a10 : assert property (p10);
```

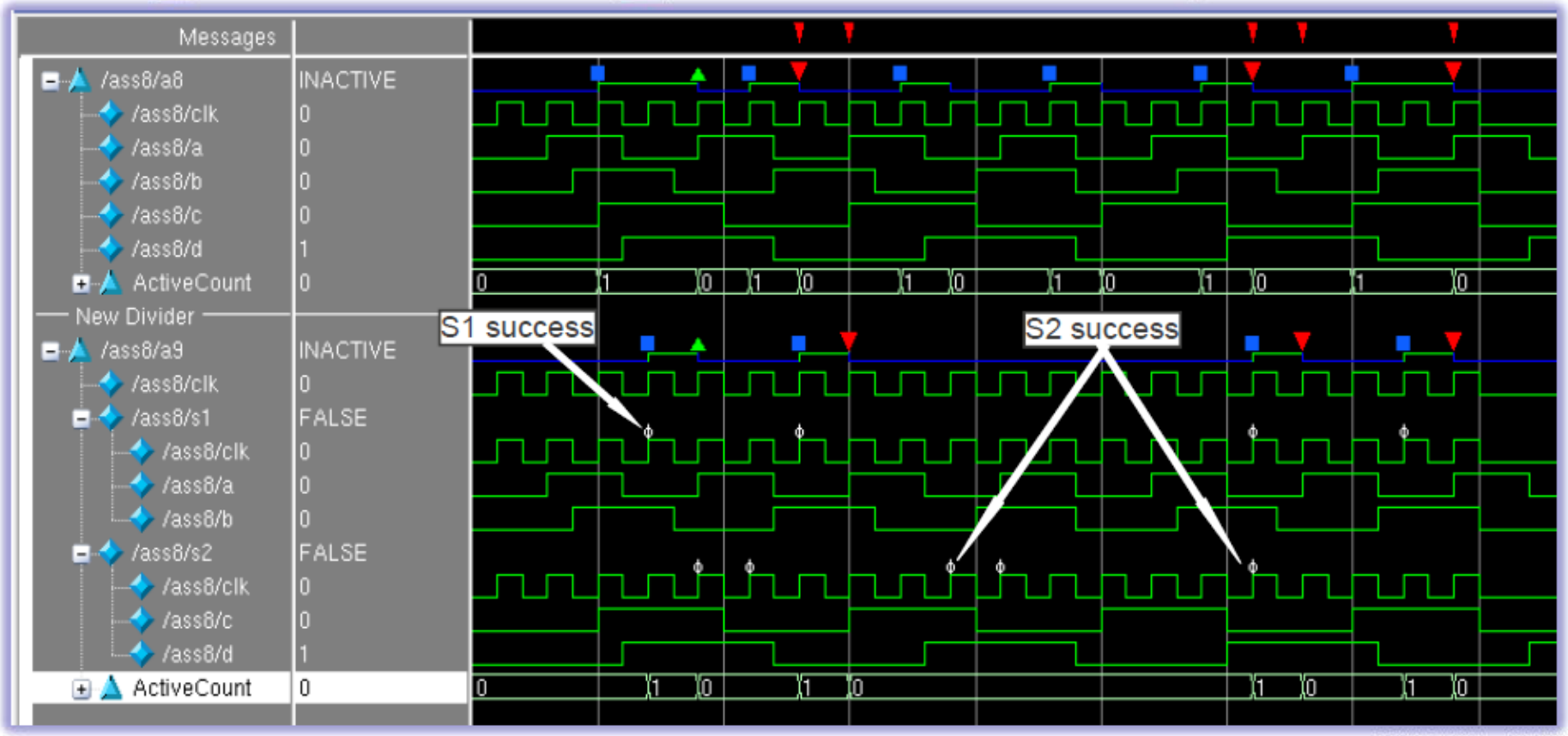


```
sequence s1;
    @(posedge clk) a ##1 b;
endsequence
sequence s2;
    @(posedge clk) c ##1 d;
endsequence
```

```
property p8; @(posedge clk) s1 |-> s2; endproperty
a8 : assert property (p8);
```

```
property p9; @(posedge clk) s1.ended |=> s2.ended;
endproperty
a9 : assert property (p9);
```

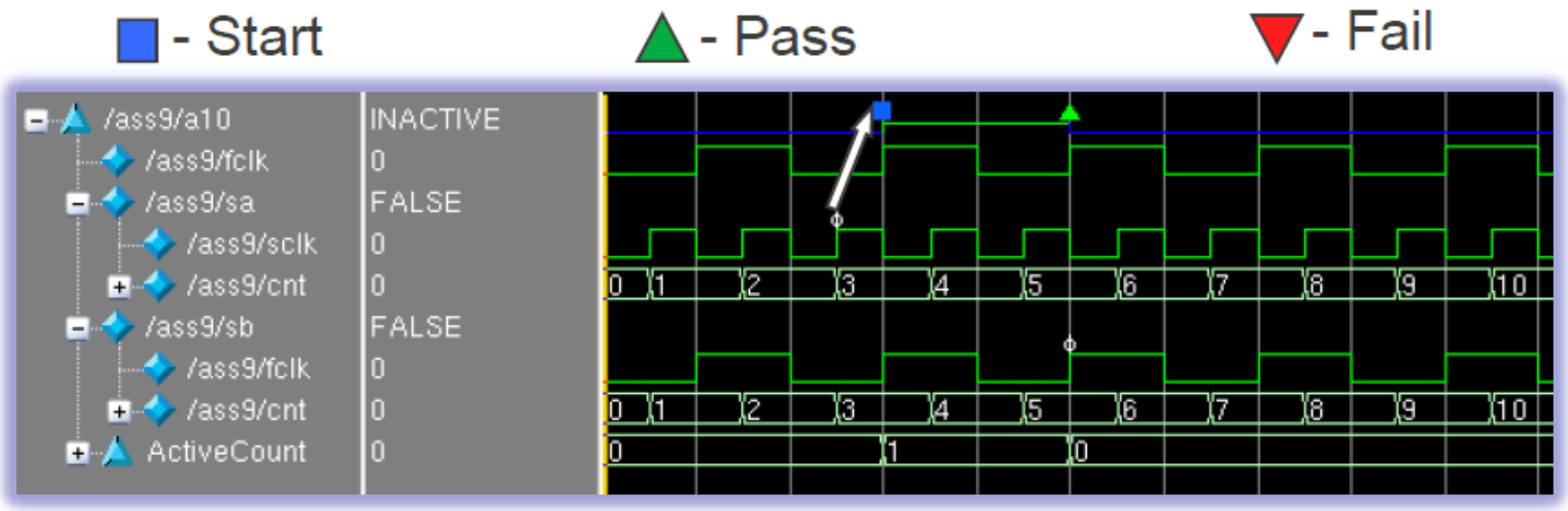
■ - Start ▲ - Pass ▼ - Fail



.matched operator

- .matched obezbeđuje sinhronizaciju između taktova
- čuva rezultate prethodnog semplovanja, sve do nailaska narednog trenutka semplovanja ciljanog takta
- koristi se .ended da se objedine sekvence u jednom takt domenu
- .matched se koristi da se objedine sekvence u različitim takt domenima

```
sequence sa; @(posedge sclk) (cnt ==1) ##1 (cnt ==2); endsequence
sequence sb; @(posedge fclk) (cnt ==3) ##1 (cnt ==5); endsequence
property p10; @(posedge fclk) sa.matched |=>  sb.ended; endproperty
a10 : assert property (p10);
```



Ponavljanja - Repetitions

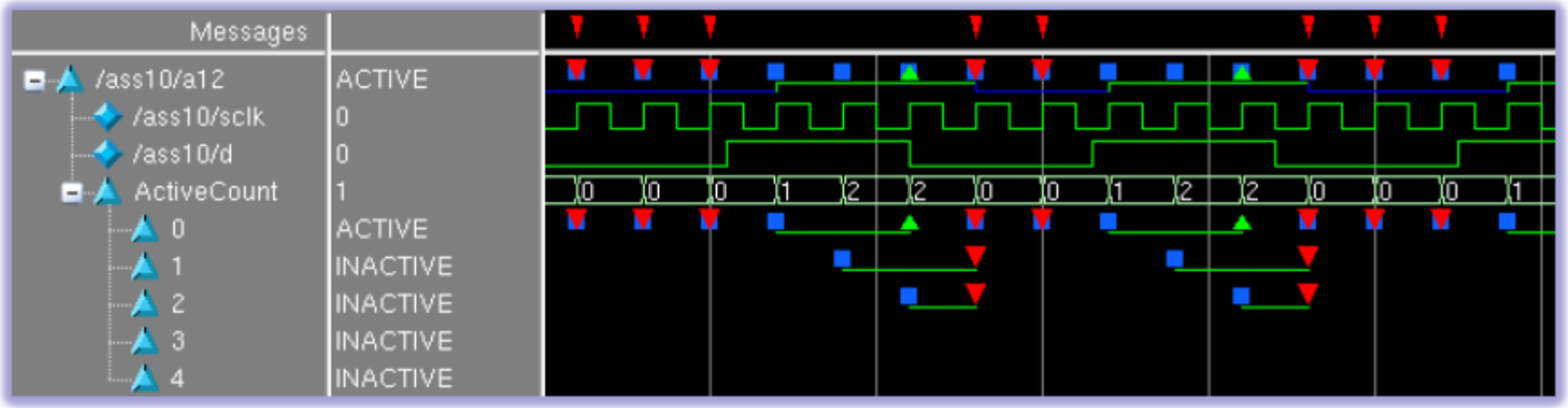
- Iterativni broj ponavljanja može biti specificiran kao tačan broj ili unutra opsega.
- Moguće su tri vrste ponavljanja
  - Uzastopno ponavljanje
  - goto ponavljanje
  - ne-uzastopno ponavaljanje

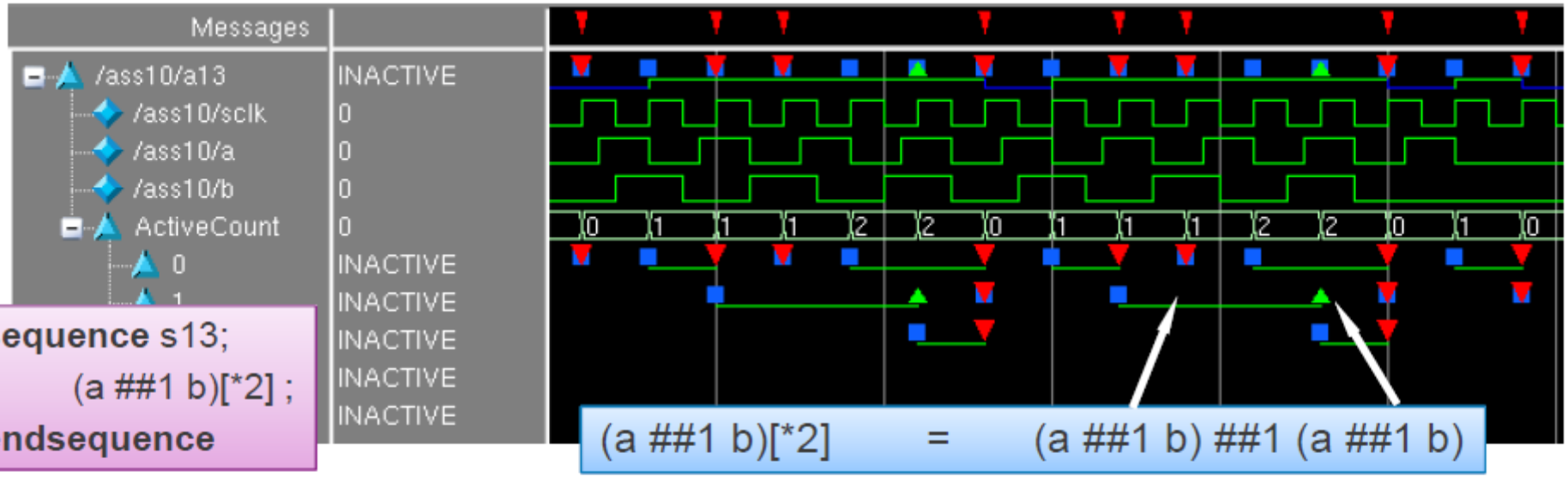
- Ponavaljanje sekvence: name[*number_of_events]
- d[*3] znači, d treba da bude visok tokom 3 semplovanja

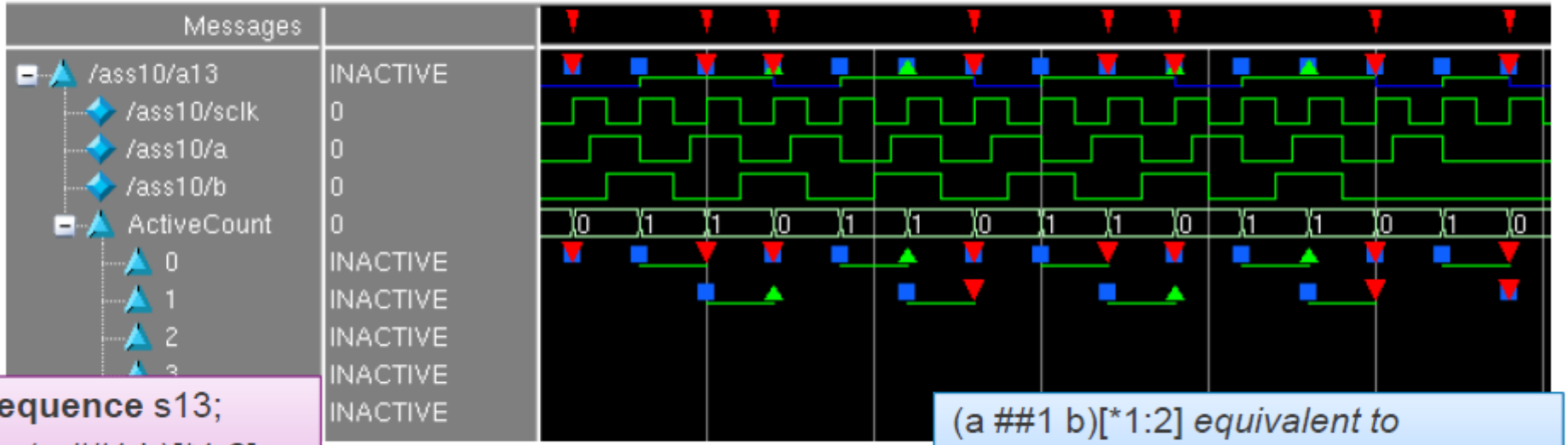
```
sequence s12;
    d [*3];
endsequence
property p12;
    @(posedge sclk) s12;
endproperty
a12 : assert property (p12);
```

Implicit ##1 delay

```
sequence s12;
    d ##1 d ##1 d;
endsequence
property p12;
    @(posedge sclk) s12;
endproperty
a12 : assert property (p12);
```

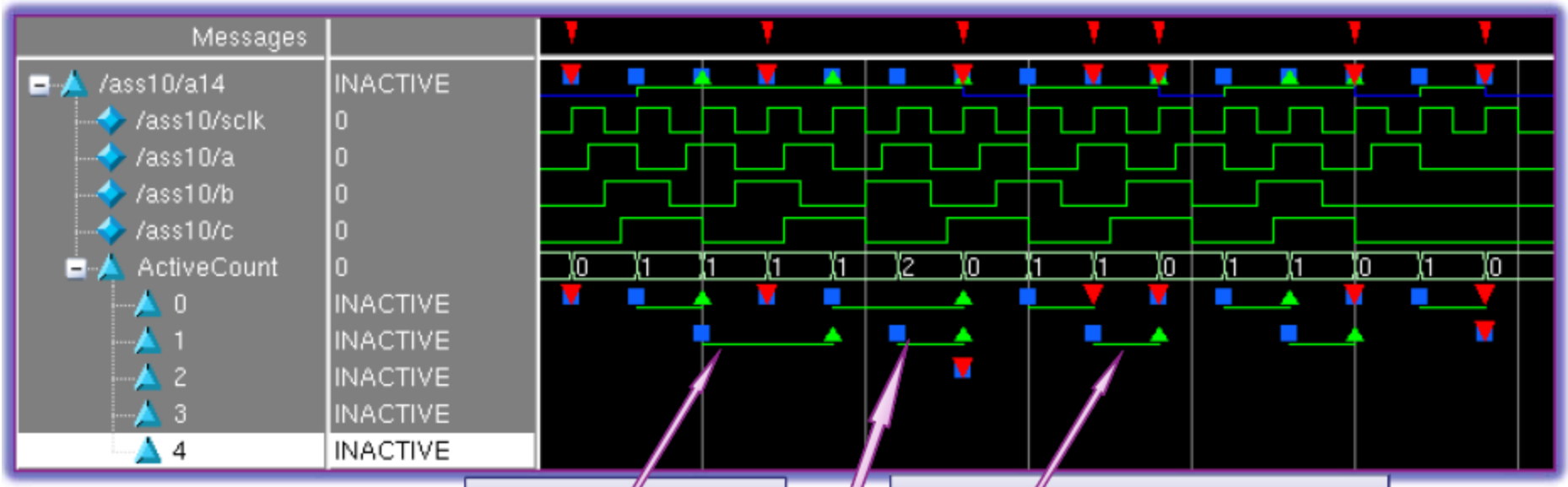



Messages
/ass10/a13
/ass10/sclk
/ass10/a
/ass10/b
ActiveCount
0
INACTIVE
0
0
0
0
INACTIVE
INACTIVE
INACTIVE
INACTIVE
INACTIVE
sequence s13;
(a ##1 b)[*2] ;
endsequence
(a ##1 b)[*2] = (a ##1 b) ##1 (a ##1 b)


Messages
/ass10/a13
/ass10/sclk
/ass10/a
/ass10/b
ActiveCount
0
1
2
INACTIVE
0
0
0
0
INACTIVE
INACTIVE
INACTIVE
INACTIVE
INACTIVE
sequence s13;
(a ##1 b)[*1:2] ;
endsequence
Range specifies [*min:max]
(a ##1 b)[*1:2] equivalent to
(a ##1 b) or
(a ##1 b) ##1 (a ##1 b)

nula u intervalu ponavljanja

- poklapa se nula ili više pojava u sekvenci
- poklapanje čak i ako se sekvenca ne pojavi
- primer: a ##1 b[*0:2] ##1 c



a ##1 b[*0:2] ##1 c equivalent to

a | c → no b

a | b | c → one b

a | b | b | c → two b

Ponavljanje sa goto

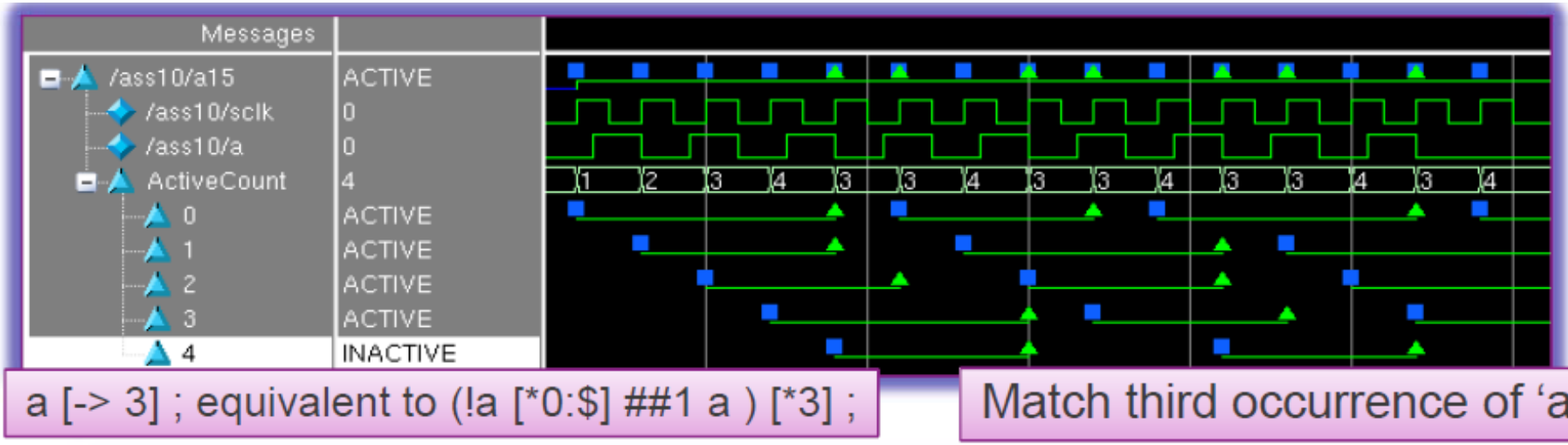
- Go to ponavljanje specificira konačan broj iterativnog ponavljanja bulovog izraza sa kašnjenjem od jednog ili više taktova

Navodi se ovako:

1 => 3 [-> 3] => 5

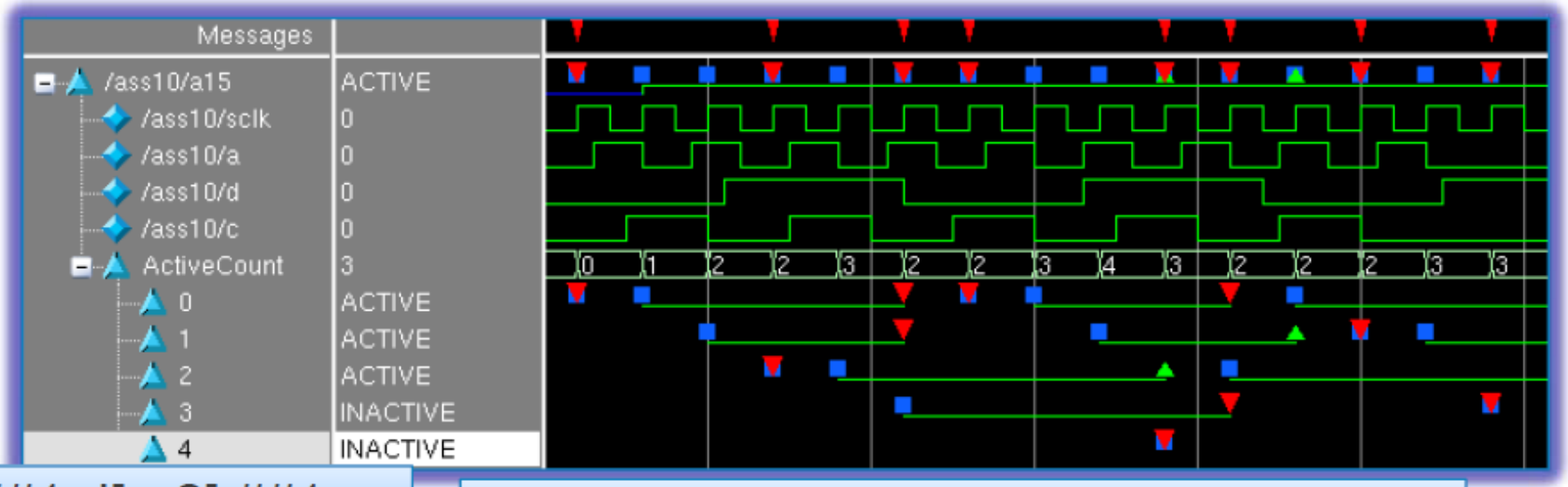
što je identično sa

1...=>3...=>3...=>3 =>5



a [-> 3] ; equivalent to (!a [*0:$] ##1 a) [*3] ;

Match third occurrence of ‘a’



a ##1 d[->2] ##1 c;

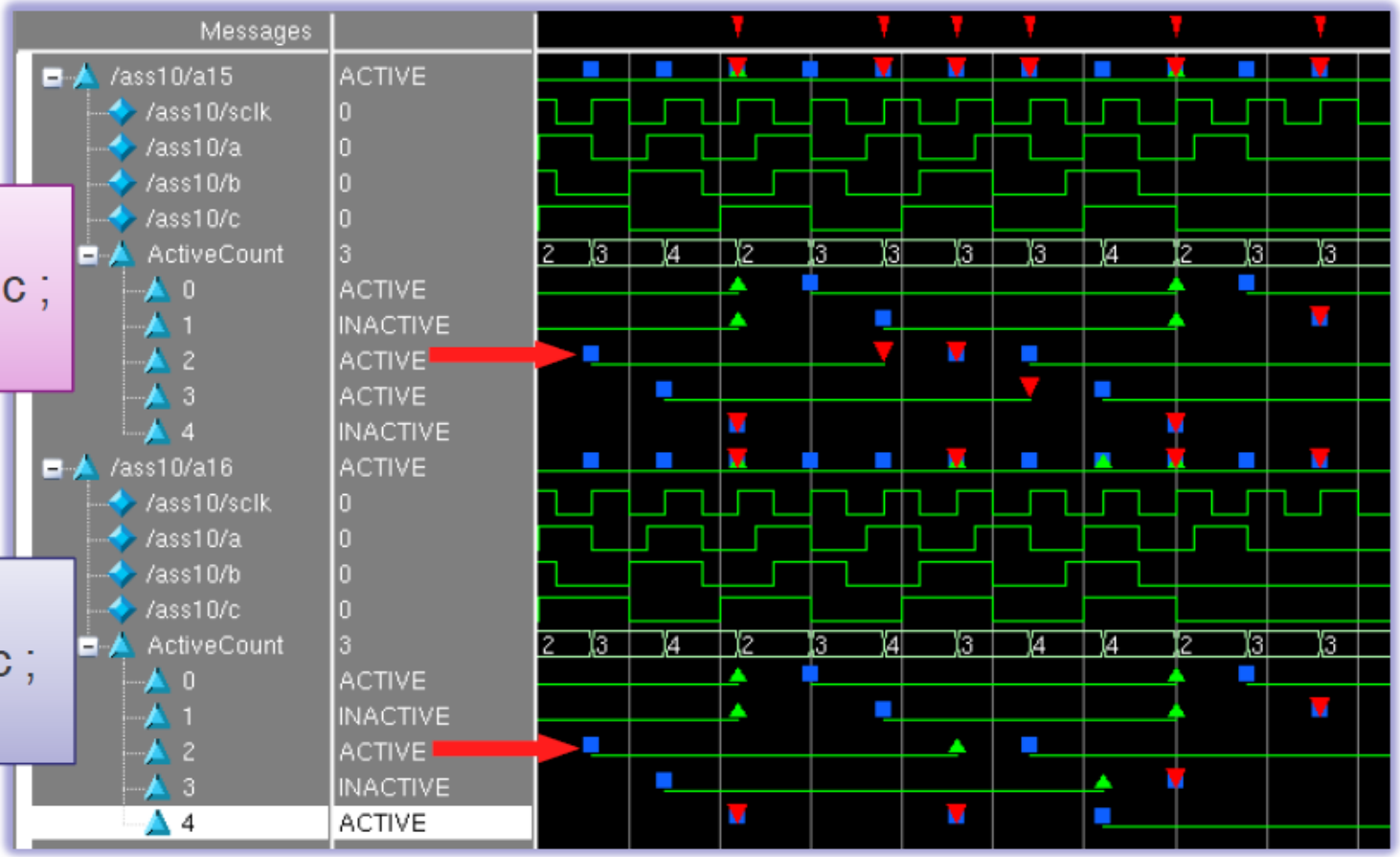
d is asserted, intermittently or consecutively

Ne uzastopno ponavljanje

- Go-to ponavljanje se završava u istom taktu
- Ne uzastopno ponavljanje može da čeka do pojave poklapanja

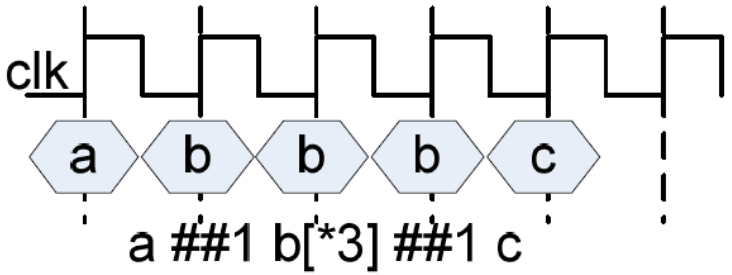
```
sequence s15;
a ##1 b[->2] ##1 c ;
endsequence
```

```
sequence s16;
a ##1 b[=2] ##1 c ;
endsequence
```

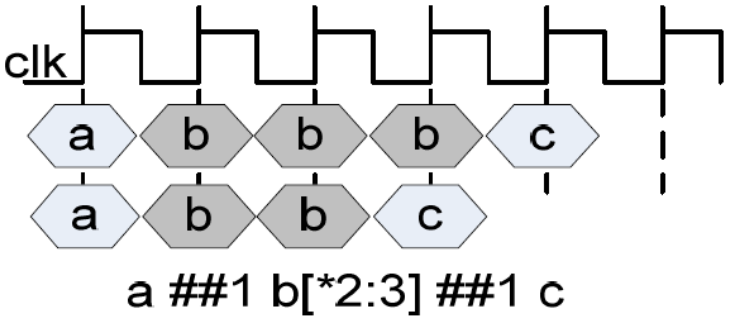


Ponavljanje osvrt

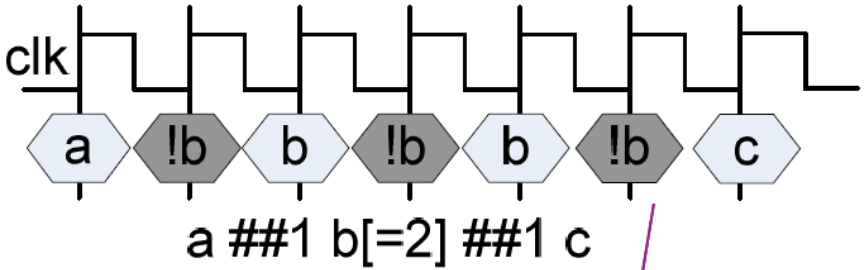
Consecutive repetition



Sequence range repetition

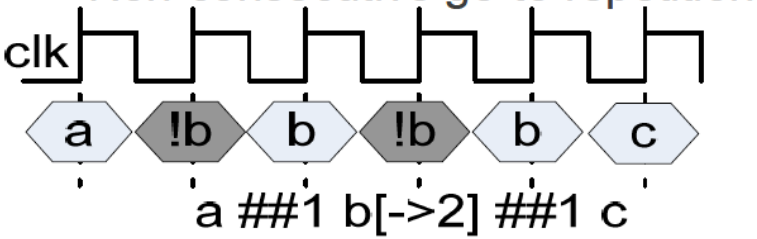


Non-consecutive Counting repetition



Matching doesn't have to end with a expr

Non-consecutive go-to repetition



Matching ends with a expr

Primer sa raspoređivanjem sekvence

```
property p1;
    @(posedge clk) (X ##1 Y) |=> Z
endproperty
```