



## ■ Sekvencijalni i paralelni blokovi

- Više iskaza se grupišu u blok pomoću iskaza za grupisanje, da bi se ponašali kao jedan iskaz
- U dosadašnjim primjerima smo koristili ključne riječi **begin** i **end** za grupisanje iskaza
- Radilo se o **sekvencijalnim blokovima**: iskazi u bloku se *izvršavaju* jedan **poslije** drugoga
  - Iskazi u sekvencijalnom bloku se obrađuju onim redom kojim su navedeni
  - Iskaz se *izvršava* tek nakon što prethodni iskaz završi *izvršavanje* (osim kod neblokirajućeg dodjeljivanja sa intra-assignment vremenskom kontrolom)
  - Ako se specificira kašnjenje ili događaj, posmatra se u odnosu na vremenski trenutak u kojem je prethodni iskaz završio sa *izvršavanjem*

## ■ Sekvencijalni blok – primjer

```
reg x, y;      // sekvencijalni blok bez kašnjenja
```

```
reg [1:0] z, w;
```

```
Initial
```

```
begin
```

```
    x = 1'b0;
```

```
    y = 1'b1;
```

```
    z = {x, y};
```

```
    w = {y, x};
```

```
end    // krajnje vrijednosti su x = 0, y = 1, z = 1, w = 2, u time=0
```

```
reg x, y;      // sekvencijalni blok sa kašnjenjem
```

```
reg [1:0] z, w;
```

```
initial
```

```
begin
```

```
    x = 1'b0;    // završava se u trenutku 0
```

```
    #5 y = 1'b1;    // završava se u trenutku 5
```

```
    #10 z = {x, y};    // završava se u trenutku 15
```

```
    #20 w = {y, x};    // završava se u trenutku 35
```

```
end    // iste krajnje vrijednosti samo u drugim vremenskim trenucima
```



## ■ Paralelni blokovi

- Specificiraju se ključnim riječima **fork** i **join**
- Imaju sljedeće karakteristike:
  - Iskazi u paralelnom bloku se *izvršavaju* konkurentno
  - Redoslijed se kontroliše kašnjenjima ili događajima, pridruženim iskazima
  - Ako je specificiran događaj ili kašnjenje, odnosi se na trenutak *ulaska* u blok
- Uočiti glavnu razliku između sekvencijalnog i paralelnog bloka: svi iskazi u paralelnom bloku se “*startuju*” u trenutku ulaska u blok => **redoslijed navođenja iskaza u paralelnom bloku nije važan**



## ■ Paralelni blok – primjer

- Konvertovati prethodni primjer sekvencijalnog bloka u paralelni:

```
reg x, y;      // sekvencijalni blok bez kašnjenja
reg [1:0] z, w;
initial
  fork
    x = 1'b0;      // završava se u trenutku 0
    #5 y = 1'b1;    // završava se u trenutku 5
    #10 z = {x, y}; // završava se u trenutku 10
    #20 w = {y, x}; // završava se u trenutku 20
  join // blok se završava u trenutku 20 umjesto u trenutku 35
```

## ■ Paralelni blokovi – nastavak

- Sa paralelnim blokovima se mora biti pažljiv: *race condition*
- Prvi primjer sa sekvencijalnim blokom, ako se pretvori u paralelni blok:

```
reg x, y;  
reg [1:0] z, w;  
initial  
    fork  
        x = 1'b0;  
        y = 1'b1;  
        z = {x, y};  
        w = {y, x};  
    join
```

- Javlja se *race condition*: svi iskazi započinju u trenutku *time=0*; redoslijed izvršavanja nije poznat; ako **x** i **y** dobiju vrijednosti prije **z** i **w**, onda će biti **z=1** i **w=2**; ako ih dobiju poslije, biće **z=w=2'bxx**
- Rezultat za **z** i **w** zavisi od implementacije simulatora! (nije determinisan)



## ■ Paralelni blokovi – zaključak

- Ključna riječ **fork** se može posmatrati kao račvanje jednog toka u više nezavisnih tokova
- Ključna riječ **join** se može posmatrati kao spajanje (udruživanje) nezavisnih tokova u jedan zajednički tok
- Nezavisni tokovi se obavljaju konkurentno

## ■ Osobine blokova

- Ugnježđivanje blokova (sekvencijalni i paralelni se mogu kombinovati):

```
`timescale 1ns / 1ps
module nested_bloks;
reg x, y;
reg [1:0] z, w;
initial
begin
    $monitor($time, " x=%b, y=%b, z=%b, w=%b", x, y, z, w);
    x=1'b0;
    fork
        #5 y=1'b1;
        #10 z={x, y};
    join
        #20 w={y, x};
end
endmodule
```

```
0 x=0, y=x, z=xx, w=xx
5 x=0, y=1, z=xx, w=xx
10 x=0, y=1, z=01, w=xx
30 x=0, y=1, z=01, w=10
```



## ■ Osobine blokova – nastavak

- **Imenovanje blokova** (bloku se može dati ime):
  - U imenovanom bloku se mogu deklarirati lokalne promjenljive
  - Imenovani blok je dio hijerarhije dizajna: može se pristupiti promjenljivima po *punom* imenu
  - Imenovani blokovi se mogu *isključiti* (zaustaviti njihov rad)

module top;

initial

begin: blok1 // sekvencijalni blok nazvan *blok1*

integer i; // cjelobrojna prom. *i* je lokalna u bloku *blok1*

... // može joj se pristupiti kao top.blok1.i

end

initial

fork: blok2 // paralelni blok nazvan *blok2*

reg i; // registarska prom. *i* je lokalna u bloku *blok2*

... // može joj se pristupiti kao top.blok2.i

join



## ■ Osobine blokova – nastavak

- **Isključivanje imenovanih blokova** (imenovani blok se može isključiti)
- Blok se isključuje pomoću ključne riječi **disable**
- Koristi se da se izađe iz petlje, servisira pojava greške, kontroliše izvršavanje dijela koda na osnovu nekog kontrolnog signala, ...
- Kad se blok isključi, *izvršavanje* se preusmjerava na iskaz koji se nalazi neposredno nakon bloka
- Slično kao **break** za izlazak iz petlje u programskom jeziku C – samo što se **break** odnosi na tekuću petlju, a **disable** omogućava isključivanje bilo kojeg imenovanog bloka unutar dizajna
- Prisjetimo se primjera sa traženjem prvog bita koji ima vrijednost 1 u vektoru (koristili smo **promjenljivu continue kao semafor**)



## ■ Može bez uvođenja *semafora*

```
// Nadji prvi bit sa log. 1 u vektorskoj promjenljivoj
`define TRUE 1'b1;
`define FALSE 1'b0;
reg [15:0] flag;
integer i; // brojac
reg continue; // za prekid petlje
initial
begin
    flag = 16'b 0010_0000_0000_0000; i = 0; continue = `TRUE;
    while((i < 16) && continue)
    begin
        if (flag[i])
        begin
            $display("Nadjena log. jedinica na bitu broj %d", i);
            continue = `FALSE;
        end
        i = i + 1;
    end
end
```



## ■ Osobine blokova – nastavak

```
// Nadji prvi bit sa log. 1 u vektorskoj promjenljivoj
reg [15:0] flag;
integer i; // brojac
initial
begin
    flag = 16'b 0010_0000_0000_0000; i = 0;
    begin: petlja
        while(i < 16)
            begin
                if (flag[i])
                    begin
                        $display("Nadjena log. jedinica na bitu broj %d", i);
                        disable petlja;
                    end
                i = i + 1;
            end
        end
    end
end
```



## ■ Primjer 1 (*behavioral* dizajn) – MUX 4/1

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);  
    output out;  
    input i0, i1, i2, i3;  
    input s1, s0;  
    reg out;  
    // preračunati signal out ako se promijeni bilo koji od ulaznih signala  
    always @(s1 or s0 or i0 or i1 or i2 or i3)  
        begin  
            case ({s1, s0})  
                2'b00: out = i0;  
                2'b01: out = i1;  
                2'b10: out = i2;  
                2'b11: out = i3;  
                default: out = 1'bx;  
            endcase  
        end  
endmodule
```



## ■ Primjer 2 (*behavioral* dizajn) – brojač (4 bita)

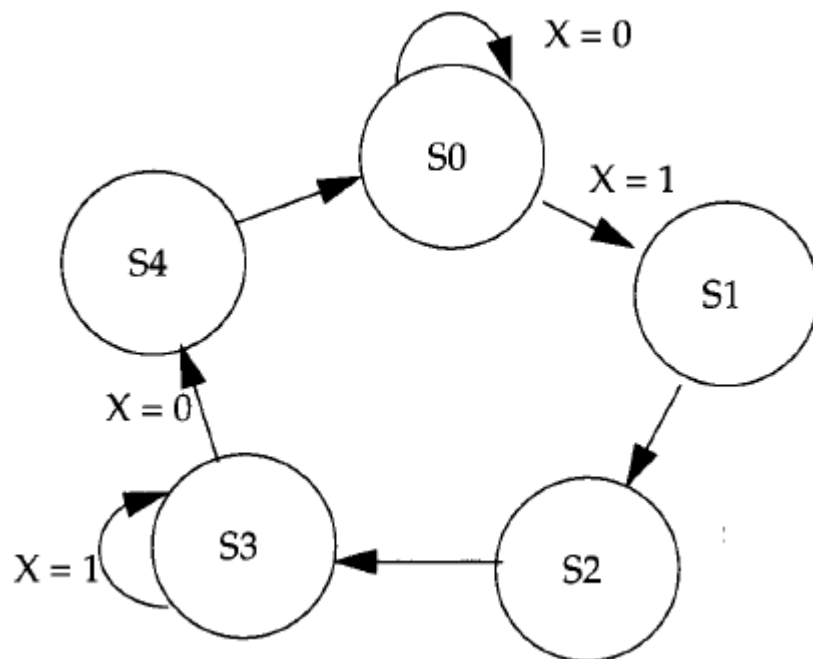
```
module counter(Q, clock, clear);  
output [3:0] Q;  
input clock, clear;  
reg [3:0] Q;  
always @(posedge clear or negedge clock)  
begin  
    if (clear)  
        Q = 4'd0;  
    else  
        Q = (Q + 1) % 16;  
    end  
endmodule
```



## ■ **Primjer 3 (*behavioral* dizajn) – kontroler za semafor**

- Ukrštanje glavnog i sporednog puta (put sa jako malom frekvencijom saobraćaja)
- Glavni put ima prioritet (u startu je zeleno svijetlo)
- Zeleno svijetlo na sporednom putu traje samo toliko da se propuste vozila koja se tamo nalaze
- Čim tamo nema više vozila pali se žuto, pa crveno svijetlo, a na glavnom putu se pali zeleno svijetlo
- Na sporednom putu se nalazi senzor:  $X=1$  ako ima vozila na sporednom putu;  $X=0$  ako tamo nema vozila
- Problem se rješava upotrebom automata (Moor)

## ■ Primjer 3 (kontroler za semafor) – nastavak



Prelazak iz stanja S1 u S2, iz stanja S2 u S3 i iz stanja S4 u S0 se mora obavljati sa nekim kašnjenjem

- Stanje S0: glavni – zeleno; sporedni – crveno
- Stanje S1: glavni – žuto; sporedni – crveno
- Stanje S2: glavni – crveno; sporedni – crveno
- Stanje S3: glavni – crveno; sporedni – zeleno
- Stanje S4: glavni – crveno; sporedni – žuto



## ■ Primjer 3 (kontroler za semafor) – nastavak

- Umjesto brojeva treba koristiti razumljive oznake:

```
`define TRUE 1'b1
`define FALSE 1'b0
// definicije signala
`define CRVENO 2'd0
`define ZUTO 2'd1
`define ZELENO 2'd2
// definicije stanja: glavni sporedni
`define S0 3'd0 // zeleno crveno
`define S1 3'd1 // zuto crveno
`define S2 3'd2 // crveno crveno
`define S3 3'd3 // crveno zeleno
`define S4 3'd4 // crveno zuto
// kašnjenja
`define ZUTO_CRVENO 3
`define CRVENO_ZELENO 2
```



## ■ Primjer 3 (kontroler za semafor) – nastavak

```
module semafor(glavni, sporedni, X, clock, clear);  
output [1:0] glavni, sporedni; // 3 moguća svjetla (C,Ž,Z)  
reg [1:0] glavni, sporedni;  
input X; // ako je TRUE ima vozila na sporednom putu  
input clock, clear;
```

```
// pomocne promjenljive (za stanja automata)  
reg [2:0] stanje;  
reg [2:0] sljedece_stanje;
```

```
// semafor počinje u stanju S0  
initial // inicijalizacija  
begin  
    stanje = `S0;  
    sljedece_stanje = `S0;  
    glavni = `ZELENO;  
    sporedni = `CRVENO;  
end ...
```

## ■ Primjer 3 (kontroler za semafor) – nastavak

// automat

always @(stanje or clear or X)

begin

if (clear)

sljedece\_stanje = `S0;

else

case (stanje)

`S0: if(X) sljedece\_stanje = `S1;  
else sljedece\_stanje = `S0;

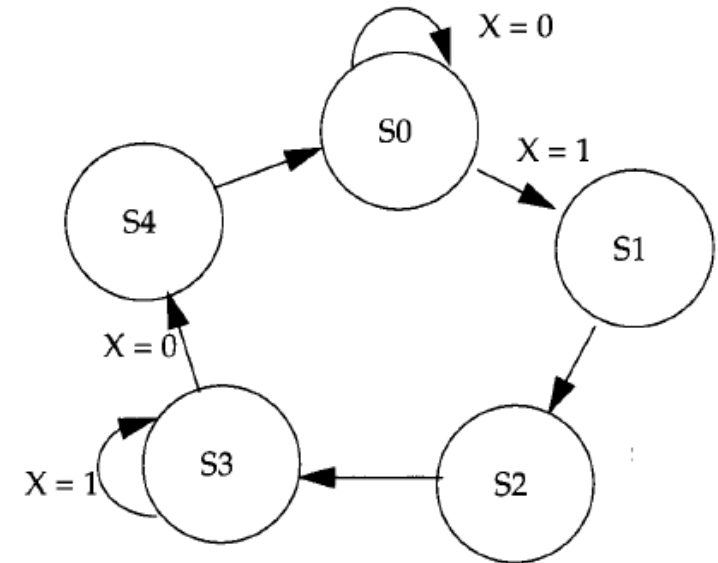
`S1: begin // sačekaj nekoliko pozitivnih ivica takta  
repeat(`ZUTO\_CRVENO) @(posedge clock);  
sljedece\_stanje = `S2;

end

`S2: begin // sačekaj nekoliko pozitivnih ivica takta  
repeat(`CRVENO\_ZELENO) @(posedge clock);  
sljedece\_stanje = `S3;

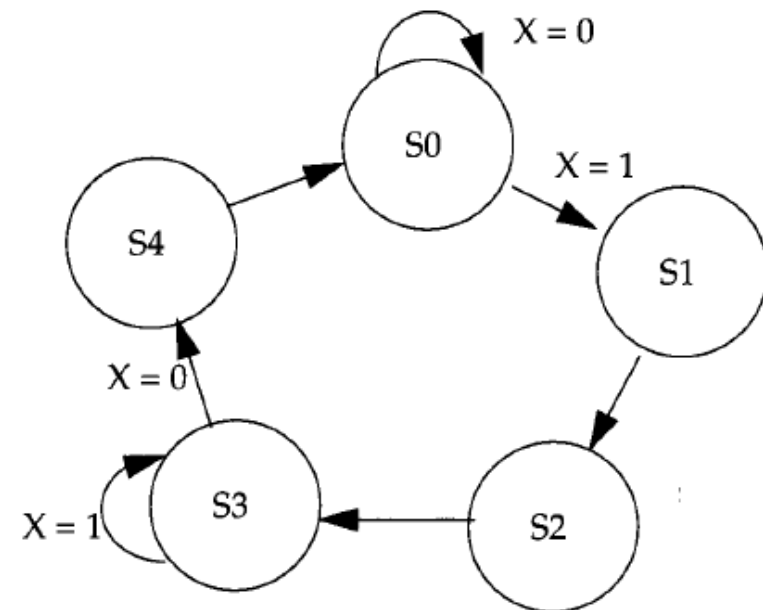
end

`S3: ...



## ■ Primjer 3 (kontroler za semafor) – nastavak

```
`S3:  if(X) sljedece_stanje = `S3;  
      else sljedece_stanje = `S4;  
`S4:  begin // sačekaj nekoliko pozitivnih ivica takta  
        repeat(`ZUTO_CRVENO) @(posedge clock) ;  
        sljedece_stanje = `S0;  
      end  
default: sljedece_stanje = `S0;  
endcase  
end
```



## ■ Primjer 3 (kontroler za semafor) – nastavak

// promjena stanja se vrši na pozitivnoj ivici takta

```
always @(posedge clock)  
    stanje = sljedece_stanje;
```

// Izračunati vrijednost svjetlosnih signala

```
always @(stanje)
```

```
begin
```

```
    case(stanje)
```

```
        `S0:    begin
```

```
            glavni = `ZELENO;  
            sporedni = `CRVENO;
```

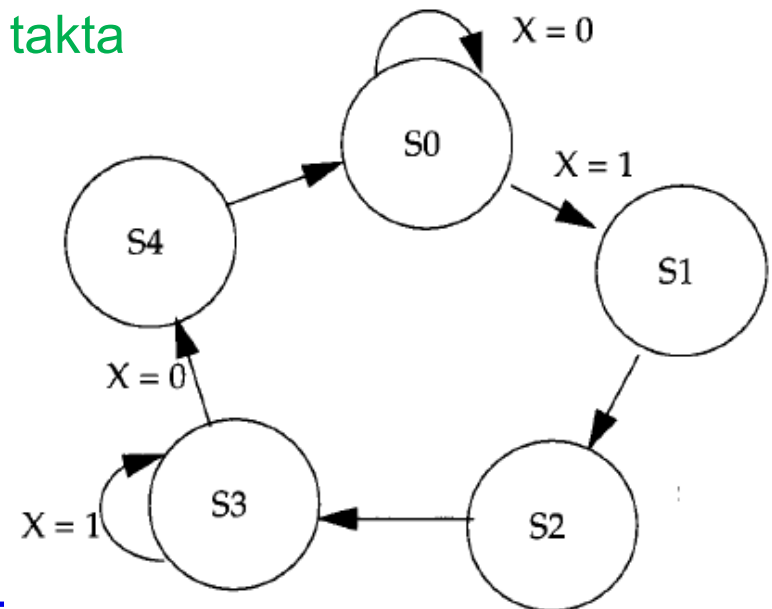
```
        end
```

```
        `S1:    begin
```

```
            glavni = `ZUTO;  
            sporedni = `CRVENO;
```

```
        end
```

```
        `S2:    ...
```



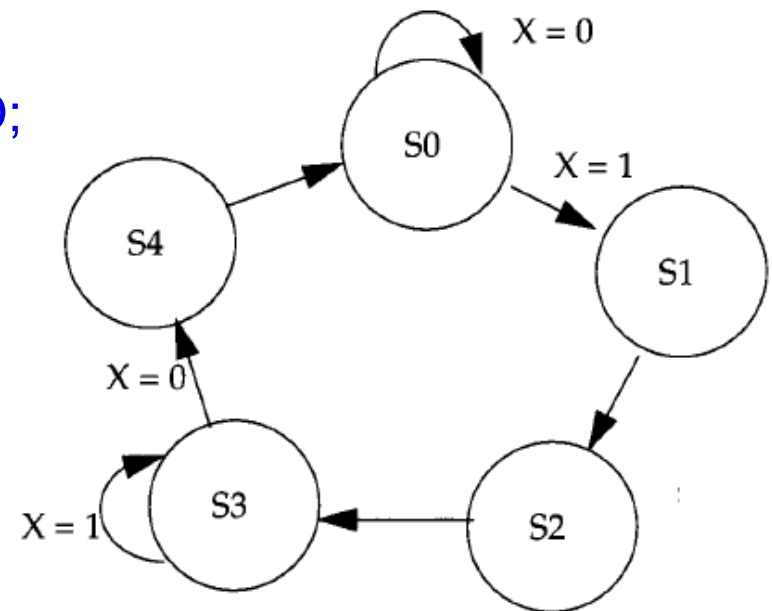
## ■ Primjer 3 (kontroler za semafor) – nastavak

```
`S2:    begin
        glavni = `CRVENO;
        sporedni = `CRVENO;

        end
`S3:    begin
        glavni = `CRVENO;
        sporedni = `ZELENO;

        end
`S4:    begin
        glavni = `CRVENO;
        sporedni = `ZUTO;

        end
    endcase
end
endmodule
```





## ■ Primjer 3 (kontroler za semafor) – nastavak

```
module stimulus;  
wire [1:0] GLAVNI, SPOREDNI;  
reg VOZILO_NA_SPOREDNOM;  
reg CLOCK, CLEAR;
```

```
// instanciranje kontrolera
```

```
semafor S(GLAVNI, SPOREDNI, VOZILO_NA_SPOREDNOM, CLOCK, CLEAR);
```

```
initial
```

```
    $monitor ($time, " Glavni = %b Sporedni = %b X = %b",  
              GLAVNI, SPOREDNI, VOZILO_NA_SPOREDNOM);
```

```
initial
```

```
begin
```

```
    CLOCK = `FALSE;  
    forever #5 CLOCK = ~CLOCK;
```

```
end
```

```
...
```

## ■ Primjer 3 (kontroler za semafor) – nastavak

```
initial // resetovanje kontrolera
```

```
begin
```

```
    CLEAR = `TRUE;
```

```
    repeat (5) @(negedge CLOCK);
```

```
    CLEAR = `FALSE;
```

```
end
```

```
// stimulus
```

```
initial
```

```
begin
```

```
    VOZILO_NA_SPOREDNOM = `FALSE;
```

```
    #200 VOZILO_NA_SPOREDNOM = `TRUE;
```

```
    #100 VOZILO_NA_SPOREDNOM = `FALSE;
```

```
    #200 VOZILO_NA_SPOREDNOM = `TRUE;
```

```
    #100 VOZILO_NA_SPOREDNOM = `FALSE;
```

```
    #200 VOZILO_NA_SPOREDNOM = `TRUE;
```

```
    #100 VOZILO_NA_SPOREDNOM = `FALSE;
```

```
    #100 $finish;
```

```
end
```

```
endmodule
```

```
0 Glavni = 10 Sporedni = 00 X = 0
```

```
200 Glavni = 10 Sporedni = 00 X = 1
```

```
205 Glavni = 01 Sporedni = 00 X = 1
```

```
245 Glavni = 00 Sporedni = 00 X = 1
```

```
275 Glavni = 00 Sporedni = 10 X = 1
```

```
300 Glavni = 00 Sporedni = 10 X = 0
```

```
305 Glavni = 00 Sporedni = 01 X = 0
```

```
345 Glavni = 10 Sporedni = 00 X = 0
```

```
500 Glavni = 10 Sporedni = 00 X = 1
```

```
505 Glavni = 01 Sporedni = 00 X = 1
```

```
545 Glavni = 00 Sporedni = 00 X = 1
```

```
575 Glavni = 00 Sporedni = 10 X = 1
```

```
600 Glavni = 00 Sporedni = 10 X = 0
```

```
605 Glavni = 00 Sporedni = 01 X = 0
```

```
645 Glavni = 10 Sporedni = 00 X = 0
```

```
800 Glavni = 10 Sporedni = 00 X = 1
```

```
805 Glavni = 01 Sporedni = 00 X = 1
```

```
...
```



## ■ ***Behavioral* dizajn – rezime**

- Digitalni sklop se opisuje u obliku algoritma koji taj sklop izvršava – ne mora da sadrži detalje hardverske implementacije
- Osnova modelovanja su proceduralne strukture **initial** i **always**: svi ostali iskazi se mogu naći samo u okviru ovih blokova
  - **initial** blok se *izvršava* jednom
  - **always** blok se *izvršava* neprekidno, dok se simulacija ne završi
- Vrijednosti registarskih promjenljivih se zadaju pomoću proceduralnog dodjeljivanja
  - **Blokirajuće** dodjeljivanje se mora završiti prije nego se *izvrši* sljedeći iskaz
  - **Neblokirajuće** dodjeljivanje “zakazuje” *izvršavanje* dodjeljivanja i nastavlja da procesuirati sljedeći iskaz



## ■ ***Behavioral* dizajn – rezime**

### ■ Vremenska kontrola redoslijeda izvršavanja iskaza u Verilogu:

- Delay-based
  - ❖ Regular delay
  - ❖ Zero delay
  - ❖ Intra-assignment delay
- Event-based
  - ❖ Regular event
  - ❖ Named event
  - ❖ Event OR
- Level-sensitive ([wait](#))



## ■ ***Behavioral* dizajn** – rezime

- Uslovni iskazi se modeluju sa **if** i **else**
- U slučaju višestrukih uslova, preporučuje se korišćenje **case** (**casex**, **casez**)
- Petlje se mogu realizovati na 4 načina:
  - **while**
  - **for**
  - **repeat**
  - **forever**
- Dva tipa blokova: sekvencijalni (**begin** – **end**) i paralelni (**fork** – **join**)
- Blokovi se mogu ugnijezditi i imenovati
- Ako je blok imenovan, može se isključiti sa bilo kog mjesta unutar dizajna
- Imenovani blokovi se referenciraju po hijerarhijskom imenu



## ■ Zadaci

- Realizovati multiplekser MUX 8/1 i testirati njegov rad sa svim mogućim kombinacijama ulaznih signala.
- Realizovati digitalno kolo koje će detektovati pojavu tri uzastopne logičke jedinice na svom ulazu. Detektovanje se signalizira logičkom jedinicom na izlazu kola, u trajanju jednog taktnog impulsa. Nakon detektovanja kolo prestaje sa radom. Ponovo počinje da radi nakon resetovanja posebnim ulaznim signalom. Napraviti odgovarajući stimulus koji će detaljno provjeriti funkcionisanje ovog kola.
- Realizovati digitalno kolo koje će detektovati pojavu sekvence 101 na svom ulazu. Detektovanje se signalizira logičkom jedinicom na izlazu kola, u trajanju jednog taktnog impulsa. Moguće je preklapanje sekvenci. Napraviti odgovarajući stimulus koji će detaljno provjeriti funkcionisanje ovog kola.