Uslovni iskazi

- Odlučivanje, na osnovu određenih uslova, da li će se neki iskaz(i) *izvršiti*
- Ključne riječi su **if** i **else**
- Tri su tipa uslovnih iskaza:
  - Nema **else** – iskaz(i) se *izvršava(ju)* ili se ne *izvršava(ju)*

```
if (<neki_izraz>) iskazi_za_tacan_izraz;
```

  - Ima jedno **else** – *izvršava(ju)* se ili iskaz(i) za tačan uslov ili za netačan uslov

```
if (<neki_izraz>) iskazi_za_tacan_izraz; else iskazi_za_netacan_izraz;
```

  - Ugnježdeni **if-else** – višestruki izbor, samo se jedan *izvršava*

```
if (<izraz1>) iskazi_za_tacan_izraz1;
else if (<izraz2>) iskazi_za_tacan_izraz2;
else if (<izraz3>) iskazi_za_tacan_izraz3;
else iskazi_za_sve_gornje_netacno;
```

Uslovni iskazi – nastavak

- Kod uslovnih iskaza prvo se evaluira <izraz> uz **if**
  - Ako je tačan (1 ili nenulta vrijednost) *izvršava(ju)* se iskaz(i) za tačan izraz
  - Ako je netačan (0) ili nepoznat (x ili z) *izvršava(ju)* se iskaz(i) za netačan izraz
- Iskaz (za tačno ili netačno) može biti pojedinačni iskaz ili grupa (blok) iskaza
- Blok se mora grupisati ključnim riječima **begin** i **end**
- Pojedinačni iskaz se ne mora grupisati

Uslovni iskazi – primjeri

```
// prvi tip uslovnog iskaza
if(!lock) buffer = data;
if(enable) izlaz = ulaz;

// drugi tip uslovnog iskaza
if (broj_baferovanih < MAX_DUZINA)
      begin
         bafer = data;
         broj_baferovanih = broj_baferovanih + 1;
      end
else
      $display ( “Bafer je popunjen");
```

Uslovni iskazi – primjeri

```
// treci tip uslovnog iskaza – ALU sa tri operacije
if (alu_control == 0)
   y = x + z;
else if(alu_control == 1)
   y = x - z;
else if(alu_control == 2)
   y = x * z;
else
   $display(“Neispravan ALU controlni signal");
```

Uslovni iskazi – nastavak

- Ako ima mnogo ugnježdenih **if-else-if**, kod trećeg tipa uslovnih iskaza, postaje nepregledno pratiti sve moguće uslove
- Ovakvi slučajevi se lakše zapisuju pomoću **case** direktive
- Koriste se ključne riječi **case**, **endcase** i **default**

```
case (izraz)
   varijanta1: iskaz1;
   varijanta2 : iskaz2;
   varijanta3 : iskaz3;
   default: podrazumijevani_iskaz; // kad nijedna od varijanti nije tačna
endcase
```

- iskaz1, iskaz2, ..., podrazumijevani_iskaz mogu biti pojedinačni iskazi ili blokovi iskaza uokvireni sa **begin** i **end**
- **izraz** se poredi sa **varijantama** onim redoslijedom kojim su navedene
- *Izvršava* se iskaz koji pripada prvoj varijanti koja se podudara sa izrazom

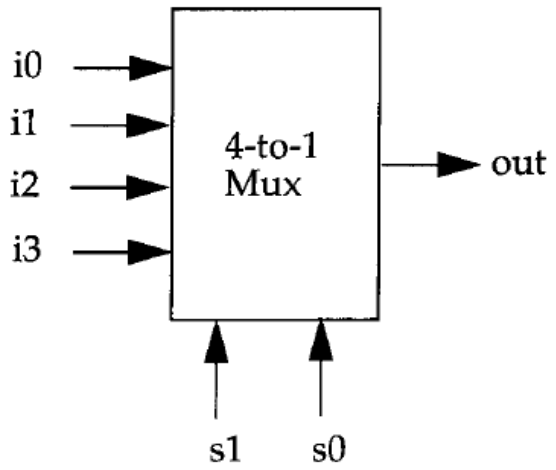
Uslovni iskazi – nastavak

- Ako se izraz ne podudara ni sa jednom varijantom, *izvršava* se iskaz koji odgovara **default**-u
- **default** je opcion – nije obavezan
- **case** iskazi se mogu ugniježdavati
- Primjer:

```
// izvršavanje iskaza zavisno od vrijednosti ALU kontrolnog signala
reg [1:0] alu_control;
...
case (alu_control)
   2'd0 : y = x + z;
   2'd1 : y = x - z;
   2'd2 : y = x * z;
   default: $display("Neispravan ALU controlni signal");
endcase
```

Uslovni iskazi – nastavak

- Primjer: Multiplekser 4/1




```
module mux4_to_1(out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3, s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
      case ({s1, s0})  // izbor se zasniva na konkatenaciji signala
         2'd0 : out = i0;
         2'd1 : out = i1;
         2'd2 : out = i2;
         2'd3 : out = i3;
         default: $display("neispravni kontrolni signali");
      endcase
endmodule
```

Uslovni iskazi – nastavak

- Porede se 0, 1, x i z vrijednosti u izrazu i u ponuđenim varijantama, bit po bit
- Ako su izraz i varijanta nejednake dužine (broja bita) kraći se dopunjava nulama
- Ako postoji više varijanti vrijednosti izraza za koje treba da se *izvrši* isti blok, mogu se staviti zajedno, odvojeni zarezima:
  - umjesto:

```
2'd0 : out = i0;
2'd1 : out = i0;
2'd2 : out = i0;
```

  - može:

```
2'd0, 2'd1, 2'd2 : out = i0;
```

Uslovni iskazi – nastavak

- Primjer: demultiplekser 1/4 sa potpunim definisanjem selekcionih signala

```
module demultiplexer1_to_4 (out0, out1, out2, out3, in, s1, s0);
... // deklaracije portova
always @(s1 or s0 or in)
      case ({s1, s0})
         2'b00 : begin out0 = in; out1 = 1'bz; out2 = 1'bz; out3 = 1'bz; end
         2'b01 : begin out0 = 1'bz; out1 = in; out2 = 1'bz; out3 = 1'bz; end
         2'b10 : begin out0 = 1'bz; out1 = 1'bz; out2 = in; out3 = 1'bz; end
         2'b11 : begin out0 = 1'bz; out1 = 1'bz; out2 = 1'bz; out3 = in; end
         2'bx0, 2'bx1, 2'bxz, 2'bxx, 2'b0x, 2'b1x, 2'bzx : // x ima prioritet nad z
            begin out0 = 1'bx; out1 = 1'bx; out2 = 1'bx; out3 = 1'bx; end
         2'bz0, 2'bz1, 2'bzz, 2'b0z, 2'b1z :
            begin out0 = 1'bz; out1 = 1'bz; out2 = 1'bz; out3 = 1'bz; end
         default: $display("Nespecificirani kontrolni signali");
      endcase
endmodule
```

Uslovni iskazi – nastavak

- Postoje dvije podvarijante **case** direktive: **casex** i **casez**
- **casez** tretira sve **z** vrijednosti u *case* alternativama ili *case* izrazu kao "bilo šta" vrijednosti; pozicije sa vrijednošću **z** se mogu predstaviti i sa **?**
- **casex** tretira sve **x** i **z** vrijednosti u *case* alternativama ili *case* izrazu kao "bilo šta" vrijednosti
- Pomoću **casex** i **casez** se mogu porediti samo pozicije koje nisu **x** ili **z** u *case* izrazu i *case* alternativama

Uslovni iskazi – nastavak

- ***Primjer***: dekodiranje bitova u četvorobitnom podatku – samo se jedan bit posmatra da se definiše sljedeće stanje, a ostali bitovi se ignorišu

```
reg [3:0] podatak;
integer sljedece_stanje;
casex (podatak)  // vrijednost x predstavlja “bilo šta” bit
   4’b1xxx : sljedece_stanje = 3;
   4’bx1xx : sljedece_stanje = 2 ;
   4’bxx1x : sljedece_stanje = 1;
   4’bxxx1 : sljedece_stanje = 0;
   default : sljedece_stanje = 0;
endcase
```

- Vrijednost podatak = 4’b10xz bi rezultiralo sa sljedece_stanje = 3

Petlje

- Postoje 4 tipa petlji u Verilogu: while, for, repeat i forever.
- Sintaksa je vrlo slična kao u programskom jeziku C
- Petlje se mogu pojaviti samo unutar initial ili always bloka
- Unutar petlji se mogu nalaziti izrazi za unošenje kašnjenja

While petlja

- Izvršava se dok uslov petlje ne postane netačan
- Ako se u petlju uđe dok je uslov netačan, petlja se neće uopšte izvršiti
- Ako je unutar petlje više iskaza, grupišu se sa begin i end

- Primjer while petlje

```
// Inkrementira promjenljivu brojac od 0 do 127
// Izlazi iz petlje za brojac==128.
// Stalno prikazuje vrijednost promjenljive brojac
integer brojac;
initial
      begin
            brojac = 0;
            while (brojac < 128)
                  begin
                     $display(“brojac = %d", brojac);
                      brojac = brojac + 1;
                  end
            $finish;
      end
```

- Primjer while petlje

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

For petlja

- Sadrži tri dijela:
  - Inicijalizaciju
  - Provjeru tačnosti uslova za prekid petlje
  - Proceduralno dodjeljivanje vrijednosti kontrolnoj promjenljivoj
- Primjer brojača sa while petljom preko for petlje:

```
integer brojac;
initial
      for (brojac=0; brojac < 128; brojac = brojac + 1)
          $display("brojac = %d", brojac);
```

- Inicijalizacija i promjena kontrolne promjenljive su uključeni u tijelo **for** petlje i ne moraju se navoditi odvojeno => kompaktnija struktura nego kod **while** petlje
- **While** petlja, sa druge strane, ima šire polje upotrebe

For petlja – nastavak

- *Primjer*: inicijalizacija niza

```
`define MAX_DUZINA 32
integer stanje[0: `MAX_DUZINA - 1];
integer i;
initial
      begin
            for(i = 0; i < 32; i = i + 2) // inicijalizacija parnih lokacija sa nulom
               stanje[i] = 0;
            for(i = 1; i < 32; i = i + 2) // inicijalizacija neparnih lokacija sa 1
               stanje[i] = 1;
      end
```

Repeat petlja

- Repeat petlja se *izvršava* konstantan broj puta
- Ne može se koristiti za petlju koja zavisi od logičkog uslova
- *Repeat* konstrukcija mora sadržati broj ponavljanja, u obliku konstante, promjenljive ili vrijednosti signala
- Ako je u pitanju promjenljiva ili signal, vrijednost se evaluira samo prilikom započinjanja petlje, a ne tokom njenog izvršavanja
- **Primjer** ranijeg brojača preko *repeat* petlje:

```
initial
begin
      brojac= 0;
      repeat(128)
            begin
                  $display(" brojac = %d", brojac);
                  brojac = brojac + 1;
            end
end
```

Repeat petlja - primjer

```
module data_buffer (data_start, data, clock);
parameter cycles = 8;   // Bafer za podatke koji ih prihvata na pozitivnoj
input data_start, clock; // ivici takta u toku 8 ciklusa nakon što primi
input [15:0] data;          // start signal
reg [15:0] buffer [0:7];
integer i;
always @(posedge clock)
      begin
            if (data_start) // start signal
                  begin
                     i = 0;
                     repeat (cycles) // smješta podatke na sljedećih 8 ivica takta
                           begin
                                 @(posedge clock) buffer[i] = data;
                                 i = i + 1;
                           end
                  end
      end
endmodule
```

Forever petlja

- Forever petlja ne sadrži nikakve izraze i izvršava se neprekidno
- Ekvivalentno sa while(1)
- Tipično se koristi zajedno sa konstrukcijama za kontrolu tajminga
- U suprotnom bi simulator beskonačno izvršavao ovaj iskaz bez napredovanja u vremenskom domenu => ostatak dizajna nikad se ne bi *izvršio*
- *Primjer*: generator takta

```
reg clock;
initial
      begin
         clock = 1'b0;
         forever  #10  clock = ~clock; // perioda je 20 vremenskih jedinica
      end
```

Forever petlja – nastavak

- *Primjer*: sinhronizovati vrijednosti dva registra na svakoj pozitivnoj ivici takta

```
reg clock;
reg X, Y;
initial
    forever @(posedge clock) X = Y;
```

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;       // paralelni blok sa kašnjenjem
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):

```
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 deklarisati 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 **nastavi** 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] vektor;
integer i; // brojac
reg nastavi; // za prekid petlje
initial
begin
   vektor = 16'b 0010_0000_0000_0000; i = 0; nastavi = `TRUE;
   while((i < 16) && nastavi)
   begin
      if (vektor[i])
      begin
                  $display("Nadjena log. jedinica na bitu broj %d", i);
                  nastavi = `FALSE;
      end
      i = i + 1;
   end
end
```

Osobine blokova – nastavak

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