Predmet: DIGITALNA ELEKTRONIKA
Predmetni nastavnik: Dr Nándor Burány

4. semestar
Broj časova: 2+2

III. DEO

PROJEKTOVANJE PRIMENOM PROGRAMABILNIH LOGIČKIH KOLA (*PLD*)

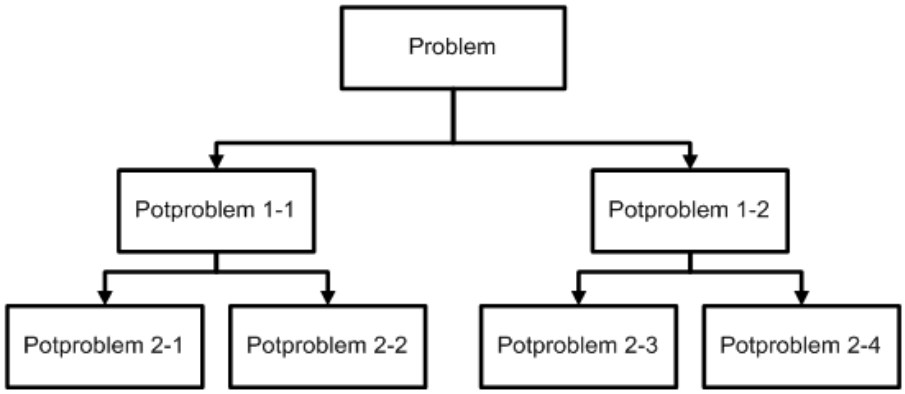
HARDVERSKI JEZIK *VERILOG*

5. Pristupi projektovanju
6. Osnovni koncepti *Verilog HDL*-a
7. Moduli i *port*-ovi
8. HDL opis na nivou logičkih kapija
9. HDL opis na nivou toka podataka
10. HDL opis na nivou ponašanja

5.a PRISTUPI PROJEKTOVANJU

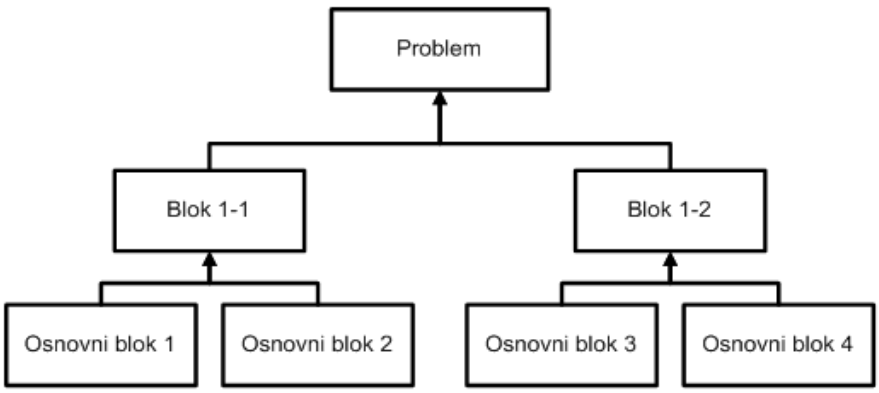
Postoje dva osnovna pristupa projektovanju digitalnih sistema:

- Top-down pristup



5.b A PRISTUPI PROJEKTOVANJU

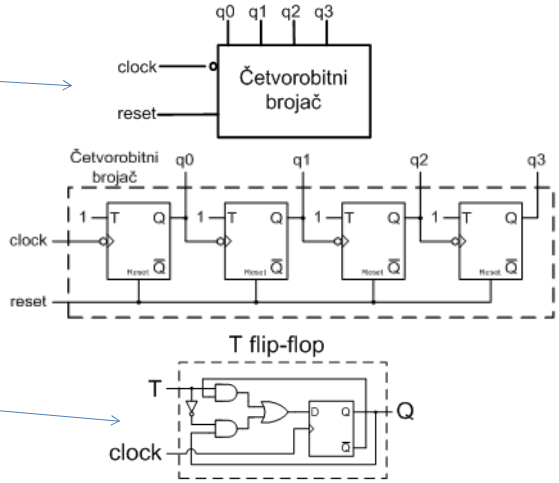
- Bottom-up pristup



- Često kombinujemo navedena dva pristupa.
- U oba slučaja postoji izvesna **hijerarhija**.

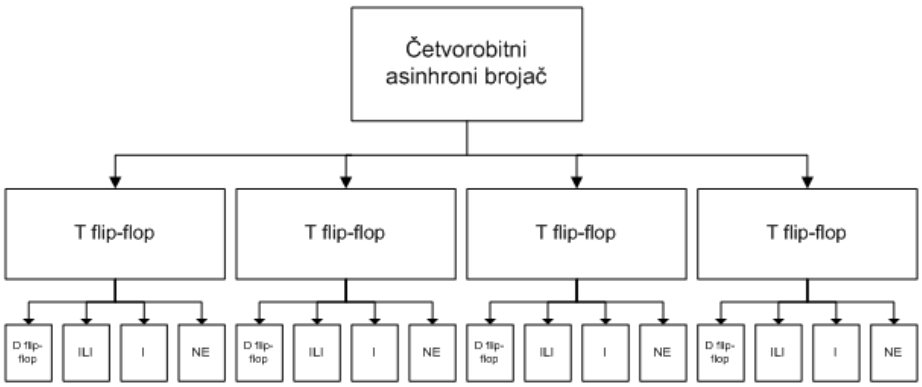
5.1.a PRIMER ZA TOP-DOWN PROJEKTOVANJE - KONSTRUISANJE ČETVOROBITNOG BROJAČA

1. Brojač (krajnji cilj)
2. Raščlanjavanje brojača na T flip-flop-ove
3. Ostvarivanje T flip-flop-ova pomoću D flip-flop-ova i logičkih kapija.



5.1.b HIJERARHIJSKA STRUKTURA ČETVOROBITNOG BROJAČA

- Struktura dobijena top-down projektovanjem:



5.2.a MODULI U VERILOG HDL-U

- Korišćenje modula omogućava hijerarhijsko projektovanje.
- Modul definiše digitalnu funkcionalnu jedinicu.
- Modul se definiše samo jednom, zatim se može koristiti u više navrata.
- Definicija modula je jedan tekstualni opis.
- Osnovna struktura modula:

```
module <ime_modula> (<lista_port-ova>);
 < telo_modula>
 endmodule
```

5.2.b MODULI U VERILOG HDL-U

Pri formiranju tekstualnog opisa koriste se sledeći **nivoi apstrakcije**:

1. Nivo ponašanja ili nivo algoritma (najviši nivo)
2. Nivo toka podataka
3. Nivo logičkih kapija
4. Nivo prekidača (najniži nivo)

- Navedena četiri nivoa se mogu slobodno **kombinovati**.
- Na osnovu tekstualnog opisa odgovarajući **softver će konstruisati digitalno** kolo koje se projektuje (logička sinteza).

5.3 INSTANCE - KORIŠĆENJE MODULA

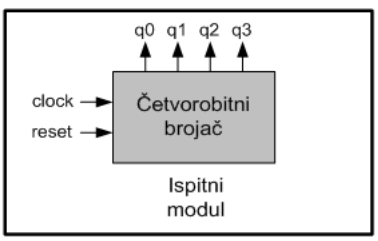
- Kreiranje konkretnog objekta na osnovu uzora (definicija modula) - to je postupak instanciranja.
- Za konkretan slučaj zadaju se veze sa okolinom.
- Modul dobije jedinstveno ime - instanca.
- Primeri instanciranja modula T_FF:

```
module brojac(q, t, clock, reset);
output [3:0] q;                    // četvorobitni izlazni signal
input t, clock, reset;             // jednobitni ulazni signali
reg t=1'b1;
T_FF tff0(q[0], 1, clock, reset);  // Instanciranje modula T_FF sa imenom tff0.
T_FF tff1(q[1], 1, q[0], reset);   // Instanciranje modula T_FF sa imenom tff1.
T_FF tff2(q[2], 1, q[1], reset);   // Instanciranje modula T_FF sa imenom tff2.
T_FF tff3(q[3], 1, q[2], reset);   // Instanciranje modula T_FF sa imenom tff3.
endmodule
```

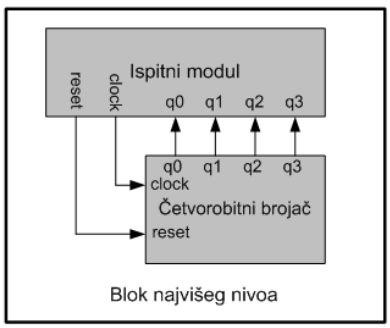
5.4 DIGITALNA SIMULACIJA HDL MODULA

- Ispituje se **ponašanje** kola bez stvarne realizacije (programiranja PLD-a).
- Ne daje 100%-nu sigurnost ali nema boljeg rešenja.
- I simulacija se obavlja **sa HDL modulima**, kao i projektovanje hardvera (međutim, takvi moduli se ne mogu sintetizovati).

1. Ispitni modul instancira modul kojim se realizuje postavljeni zadatak



2. Prazan modul instancira ispitni modul i projektovani modul



6. OSNOVNI KONCEPTI *VERILOG HDL*-A

1. Jezičke konvencije
2. Tipovi podataka i tipovi nosioca podataka

6.1 ELEMENTI I PRAVILA JEZIKA VERILOG HDL

Verilog jezičke konstrukcije se sastoje od **niza zapisa** (*token*) :

- prazno mesto (*space*),
- komentar (*comment*),
- operator (*operator*),
- broj (*number*),
- niz znakova (*string*),
- identifikator (*identifier*),
- ključna reč (*keyword*).

Softver za tumačenje Verilog HDL konstrukcija **razlikuje velika i mala slova** (*case sensitive*).

6.1.1 PRAZNA MESTA

Nemaju posebno značenje osim što **razdvajaju** druge **znakove**.

Prazna mesta se dobijaju pritiskom na sledeće tastere:

- space
- enter
- tab

Jedini slučaj kada su prazna mesta bitna je kod niza znakova (string).

6.1.2 KOMENTARI

Cilj: **dokumentovanje** HDL opisa, olakšavanje tumačenja.

Primenjuju se dve varijante:

- jednolinijski (//)
- više linijski (/* */)

```
a = b + c;      // Ovo je jednolinijski komentar

/*Ovo je višelinijski
 komentar*/
```

6.1.3 OPERATORI

- Podela: unarni, binarni, ternarni operatori.
- Primeri:

```
a = ~b;              // ~ je unarni operator. b je operand.
a = b && c;          // && je binarni operator. b i c su
                     // operandi.
a = b ? c : d;       // ?: je ternarni operator. b, c i d su
                     // operandi.
```

6.1.4.a BROJEVI

Načini pisanja:

- brojevi **sa** veličinom:

<veličina>'<osnova_brojnog_sistema><broj>

```
3'b101       // trobitni binaran broj
16'hfa3e     // 16-bitni heksadecimalan broj
16'HFA3e     // Isti broj kao prethodni samo su neka slova
             // velika
```

- brojevi **bez** veličine:

< osnova_brojnog_sistema ><broj> ili samo **<broj>**

```
'o7651       // 32-bitni oktalni broj
'hAB3        // 32-bitni heksadecimalan broj
4556         // 32-bitni decimalan broj
```

6.1.4.b BROJEVI

Specijalni slučajevi kod pisanja brojeva:

- **x** (nepoznato) i **z** (stanje velike impedanse) među ciframa broja (pojavljuju se pri opisivanju digitalnih sistema)

```
12'h13x     // 12-bitni heksadecimalni broj. Vrednosti četiri
            // najmanje značajna bita su nepoznate.
6'hx        // 6-bitni heksadecimalni broj nepoznate vrednosti.
32'bz       // 32-bitni binaran broj. Sve binarne pozicije su u
            // stanju visoke impedanse.
```

- pisanje **negativnih** brojeva

```
-8'hA1      // 8-bitni negativan broj
8'h-A1      // Pogrešno zadavanje negativnog broja
```

6.1.5 NIZOVI ZNAKOVA (STRING)

- **Skup znakova** između dva znaka navođenja.
- Ne može da bude duži od jednog reda.
- Koriste se za opisivanje simulacija - davanje povratnih informacija.
- Primer:

"Ovo je jedan niz znakova."

6.1.6 IDENTIFIKATORI

- **Nazivi**, imena objekata (modul, operand...).
- Služe za identifikaciju objekata pri njihovom pozivanju.
- Sadrže: slova, brojeve i znakove _ i $.
- Početni znak ne može biti broj ili $.

6.1.6 KLJUČNE REČI

- Specijalni podskup identifikatora.
- Služe za **definisanje jezičkih konstrukcija**.
- Uvek se pišu **malim slovima**.
- Primeri:

```
reg broj;          // reg je ključna reč; broj je identifikator
input Input;       // input je ključna reč; Input je identifikator
                   // input i Input nisu isti jer Verilog HDL
                   // razlikuje velika i mala slova.
```

6.2 TIPOVI PODATAKA I TIPOVI NOSIOCA PODATAKA

- net (wire, wand, wor, tri, triand, trior, trireg)
- reg
- vektor
- ceo broj
- realan broj
- niz (array) (nije niz znakova - string!)
- memorija
- parametar

U Verilog HDL-u se razmatraju četiri logička stanja:

Oznaka	Logičko stanje
0	logička nula
1	logička jedinica
x	nepoznato (neodređeno stanje)
z	stanje velike impedanse

6.2.2 NOSILAC PODATAKA TIPA **net**

- Za deklaraciju se koriste ključne reči: ***wire, wand, wor, tri, triand, trior, trireg***.
- Služi za predstavljanje logičkog stanja nekog voda ili čvora u digitalnom kolu.
- Vrednost dobije kada se na njega poveže izlaz logičkog kola.
- Sama reč *net* nije ključna reč!

```
wire c;          // Deklariše se podatak c, tipa veza
                 //
wire a, b;       // Deklarišu se podaci a i b, tipa veza
                 //
wire d = 1'b0;   // Podatak d je tipa veza i ima konstantnu
                 // vrednost nula.
```

6.2.3 NOSILAC PODATAKA TIPA **reg**

- Za deklaraciju se koristi ključna reč ***reg***.
- Memoriše logičku vrednost kao hardverski flip-flop.
- **Upisivanje** se ne vrši takt signalom već **dodelom**.
- Sinhronizacija dodela će se obraditi kasnije (10.3.2)
- Pre prvog dodeljivanja memorisana vrednost je **neodređena** (x).

```
reg reset;        // Nosilac podatka reset pamti dodeljene vrednosti.
initial           // Jezička konstrukcija koja će biti objašnjena
begin             // kasnije.
        reset = 1'b1;   // Pomoću reset signala se resetuje
                        // neko digitalno kolo.
        #100 reset = 1'b0;      // Posle 100 vremenskih jedinica
                                // reset signal se deaktivira.
end
```

6.2.4 VEKTORI

- **Višebitni** podaci.
- **Ne postoji posebna ključna reč** za njihovu deklaraciju. Vektori mogu biti tipa **net** ili ***reg***.
- **Potrebno je zadati broj bita**, u suprotnom slučaju deklariše se jednobitni podatak (skalar).

```
wire izlaz;       // izlaz je skalarni nosilac podatka tipa veza (net).
wire [7:0] led_indikator;       // 8-bitni vektorski nosilac podatka
wire [15:0] ledA, ledB;  // dva 16-bitna vektorska nosioca podataka
reg takt_signal;                // skalarni nosilac podatka
reg [0:255] mem;                // 256-bitni vektorski nosilac podatka.
```

- Umesto kompletnog vektora u raznim izrazima mogu se koristiti pojedini bitovi ili grupe bitova.

```
ledB [7];       // Označava sedmi bit podatka ledB.
ledA[1:0];      // Dva najmanje značajna bita podatka ledA.
```

6.2.5 PODACI TIPA **integer** (CELI BROJEVI)

- Ključna reč: ***integer***.
- Ima slične osobine kao podatak tipa ***reg***.
- Uglavnom se koristi za deklaraciju nosioca podataka tipa brojač.
- Integer može da ima i **negativnu vrednost** (registri čuvaju samo pozitivne podatke, negativni podaci mogu da se čuvaju u obliku komplementa dvojke).

```
integer brojac;          // Nosilac podatka brojac je tipa integer.
initial            // Ova jezička konstrukcija će se objaniti kasnije
        brojac = -1;     // Početna vrednost podatka brojac je -1.
```

6.2.6 PODACI TIPA **real** (REALNI BROJEVI)

- Ključna reč: ***real***.
- Ima slične osobine kao podatak tipa ***reg***.
- Mogu mu se dodeliti **realne brojne vrednosti**, **u decimalnom ili eksponencijalnom** obliku .

```
real alfa;               // Podatak alfa je tipa real.
initial
begin
        alfa = 4e10;     // Nosiocu podatka alfa se dodeljuje
                         // broj u eksponencijalnom obliku.
        alfa = 5.31;     // Nova dodela u decimalnom obliku.
end
integer i;               // Podatak i je definisan kao integer.
        i = alfa;        // Nosioc podatka se zaokruži na vrednost 5.
```

6.2.7 NIZ (ARRAY)

- Od podataka tipa ***reg*** i ***integer*** mogu se formirati jednodimenzionalni nizovi.
- Ne postoji posebna ključna reč, već odgovarajuća sintaksa: iza imena nosioca podataka napiše se dimenzija niza.

```
integer brojevi[0:7];          // Niz od 8 podataka tipa integer.
reg mem[31:0];                 // Niz od 32 1-bitnih podataka tipa reg.
reg [3:0] port [0:7];          // Niz od 8 port-ova.
                               // Svaki port je 4-bitni.
integer matrix [4:0] [4:0];    // Ilegalna deklaracija.
// Verilog standard iz 1995. ne podržava višedimenzionalne nizove,
// standard iz 2001. podržava.
brojevi[4];                    // Označava četvrti element niza brojevi.
port[5];                       // Peti element niza port.
```

6.2.8 MEMORIJE

- Ne postoji posebna ključna reč, već odgovarajuća sintaksa: iza imena nosioca podataka tipa ***reg*** napiše se dimenzija (isto kao kod vektora).
- Dužina memorijske reči može biti jedan ili više bita.

```
reg mem1bit[0:1023];           // Memorija mem1bit je skup 1024 1-
                               // bitnih reči
reg [7:0] membyte [0:1023];    // Memorija membyte je skup 1024
                               // 8-bitnih reči (bajta).
membyte[521];                  // Učitava se reč veličine jednog bajta
                               // čija je adresa 521 u nizu membyte.
```

6.2.9.a PARAMETRI

- *Verilog HDL* omogućava definisanje konstanti unutar modula primenom ključne reči ***parameter***.
- Korišćenjem parametara dobijemo preglednije opise hardvera i omogućava se lakše izvođenje izmena (manja mogućnost greške).
- Kod raznih instanciranja parametri mogu dobiti različite vrednosti.

```
parameter broj_dioda = 5; // Podatak broj_dioda je deklarisan
        //kao konstanta i dobio vrednost 5.
```

6.2.9.b PARAMETRI - NAČINI ZADAVANJA I NAČINI PRIMENE

- Donji primer prikazuje instanciranje osam isključivo ILI kola (XOR) (odnosno vrši se isključivo ILI operacija bit po bit na dva osmobitna podatka), kašnjenje pojedinih kapija je 10 vremenskih jedinica.
- Ako nam treba modul sa drugačijim brojem bita ili sa drugačijim kašnjenjem logičkih kapija, dovoljno je promeniti red za deklaraciju parametara (ne treba analizirati ceo modul).

```
module xorx
# (parameter width = 8, delay = 10)
(output [1:width] xout,
input [1:width] xinl, xin2);
assign #(delay) xout = xin1 ^ xin2;
endmodule
```

6.2.9.c PARAMETRI - PROMENE PRI INSTANCIRANJU - U REDOSLEDU ZADAVANJA

- U donjem primeru modul *xorx* definisan na prethodnom slajdu se instancira dva puta.
- U oba slučaja menja se broj bita i kašnjenje koje je deklarisano u definiciji modula.
- Nove vrednosti se zadaju u redosledu deklaracije parametara.

```
module overriddenParameters
(output [3:0] a1, a2);
reg [3:0] b1, c1, b2, c2;                    // ulazne vrednosti se obezbeđuju iz registara
xorx #(4, 0) a(a1, b1, c1), b(a2, b2, c2);          // width=4, delay=0
endmodule
```

6.2.9.d PARAMETRI - PROMENE PRI INSTANCIRANJU - PO IMENU

- U donjem primeru modul *xorx* definisan na prethodnom slajdu se ponovo instancira dva puta.
- U oba slučaja **menja se broj bita** i kašnjenje koje je deklarisano u definiciji modula.
- Nove vrednosti (koje važe samo pri ovim instanciranjima) se zadaju u zagradi, pored imena parametara.
- Ako ne zadamo nove vrednosti, ostaju vrednosti zadate u definiciji modula.

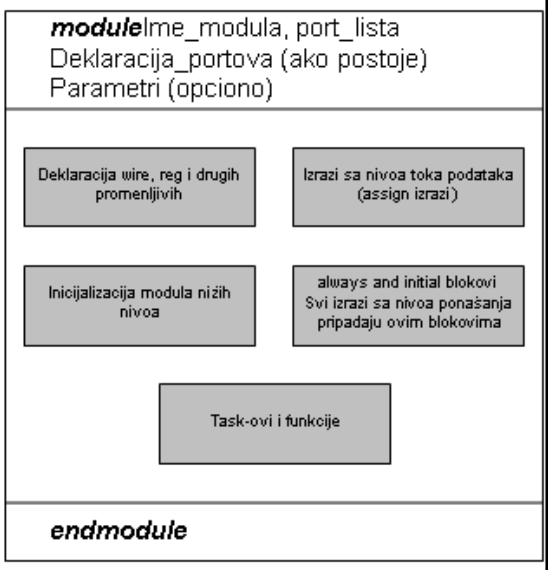
```
module overriddenParameters
(output [3:0] a1, a2);
reg [3:0] b1, c1, b2, c2;                    // ulazne vrednosti se obezbeđuju iz registara
xorx #(.width(4), .delay(0))  a(a1, b1, c1), b(a2, b2, c2);
endmodule
```

7. MODULI I PORT-OVI

1. Unutrašnja **struktura** modula:
   - ime modula
   - deklaracije
   - instanciranja
   - opisi na nivou toka podataka
   - opisi na nivou ponašanja
2. Pravila za **sprezanje** modula
   - lista port-ova
   - deklaracija port-ova
   - spajanje port-ova sa unutrašnjim nosiocima podataka
   - spajanje portova sa spoljnim signalima.

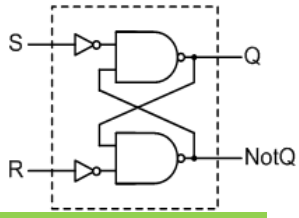
7.1.a UNUTRAŠNJA STRUKTURA MODULA

- ***module*** - svaki modul počinje sa ovom ključnom reči.
- ***endmodule*** - svaki modul se završava sa ovom ključnom reči.
- **telo modula** - opisuje funkciju koju modul treba da obavlja.



7.1.b UNUTRAŠNJA STRUKTURA MODULA

- Primer: modul za opis RS latch-a.
- Ne treba da sadrži sve moguće elemente.



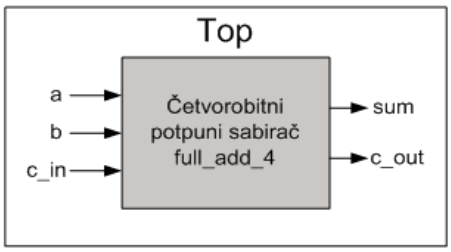

```
// Ime modula i lista port-ova
module RS_latch(Q, NotQ, R, S);
 // Deklaracija vrste port-ova
output Q, NotQ;
input R, S;
// Instanciranje modula nižih nivoa. U ovom slučaju se
// instanciraju NI kapije koje su
// osnovni gradivni elementi (primitive) koji su definsani malim
// slovima u Verilog HDL-u.
nand n1(Q, ~S, NotQ);
nand n2(NotQ, ~R, Q);
 // Kraj modula, koji je obavezan:
endmodule
```

7.2 PORT-OVI

- Port ovi igraju istu ulogu kao nožice integrisanog kola: preko njih se ostvaruje **veza unutrašnje strukture sa spoljnim kolima** (drugim modulima).
- Drugi uvid u module nemamo, ono što se vidi, vidi se preko port-ova.
- Unutrašnja struktura se ne vidi samo se detektuje funkcionalnost.
- Unutrašnja struktura se može menjati.

7.2.1 LISTA PORT-OVA

- Pri definisanju modula navode se imena port-ova.
- Može da se desi da modul nema veze sa spoljnim svetom - tada se ne stavlja lista port-ova (karakteristično za neke simulacione module).




```
module full_add_4(sum, c_out, a, b, c_in);
// Modul sa listom port-ova.
module Top;
// Modul bez liste port-ova.
```

7.2.2.a DEKLARACIJA PORT-OVA PREMA STANDARDU IZ 1995.

- Pri deklaraciji port-ova zadaje se njihov smer: ***input*** (ulaz), ***output*** (izlaz), ***inout*** (dvosmerni).

```
module full_add_4(sum, c_out, a, b, c_in);
 // Početak deklaracije port-ova
output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;
// Kraj deklaracije port-ova
 //---------------------------------
// Telo modula
//---------------------------------
endmodule
```

7.2.2.b DEKLARACIJA PORT-OVA PREMA STANDARDU IZ 1995.

- Pri deklaraciji određenog porta definiše se i istoimeni unutrašnji nosioc podatka. Podrazumeva se da će nosilac podatka biti tipa **wire**.
- Ako tip ***wire*** ne odgovara, treba ga ponovo deklarisati (deklaracijom nosioca podatka).

```
module DFF(q, d, clock, reset);
 output q;       // Pošto je port q deklarisan kao izlaz,
          //automatski mu pripada nosilac podatka tipa wire.
reg q;   // Pošto izlazna vrednost treba da se pamti,
          // podatka koji mu pripada
          // posebno se deklariše kao podatak tipa reg.
input d, clock, reset;
          // Telo modula
endmodule
```

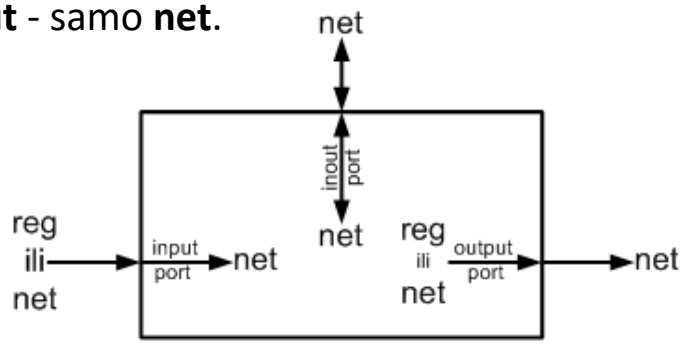
7.2.2.c DEKLARACIJA PORT-OVA PREMA STANDARDU IZ 2001.

- Deklaracije portova se mogu obaviti u listi port-ova.
- Može se zadati **veći broj deklaracija istovremeno** (u jednom redu).
- Deklaracija port-ova sa prethodnog slajda se može pisati (uprošćeno) na sledeći način:

```
module DFF(output reg q, input d, clock, reset);
          // Tipovi nosioca podataka su deklarisani u listi port-ova.
          // Tri zasebne deklaracije su sad suvišne.
          // Tu dolazi telo modula.
endmodule
```

7.2.3.a PRAVILA POVEZIVANJA PORT-OVA

- Unutrašnja deklaracija tipa podatka treba da bude u skladu sa deklaracijom smera port-a:
  - **input** - samo **net**,
  - **output** - **net** ili **reg**,
  - **inout** - samo **net**.



7.2.3.b PRAVILA POVEZIVANJA PORT-OVA

- U većini slučajeva broj bitova unutrašnjih i spoljnih vektora je isti.
- U slučaju razlike softver koji tumači opis šalje upozorenje.
- Ne moraju se svi port-ovi spajati sa okolinom.

7.2.4.a POVEZIVANJE PORT-OVA SA SIGNALIMA IZ OKRUŽENJA PREKO UREĐENE LISTE (ORDERED LIST)

- Važan je redosled, imena mogu da se razlikuju.

```
module full_add_4(sum, c_out, a, b, c_in);
output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;
        // Telo modula
endmodule

module Top;
 reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;
        full_add_4 fa0(SUM, C_OUT, A, B, C_IN);
        // Tu dolazi HDL opis koji predstavlja ispitni deo modula Top.
endmodule
```

7.2.4.b POVEZIVANJE PORT-OVA SA SIGNALIMA IZ OKRUŽENJA NA OSNOVU NJIHOVIH IMENA

- Pogodna metoda u slučaju velikog broja port-ova
- Neće se pomešati signali.
- Nepotrebni port-ovi se mogu izostaviti.

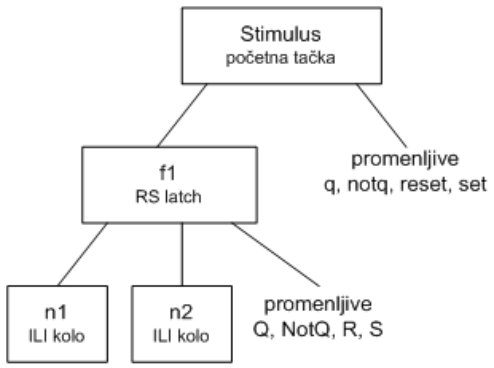
ime deklarisano u instanciranom modulu

ime koje se koristi u modulu koji vrši instanciranje

```
full_add_4 fa0(.c_out(C_OUT), .sum(SUM), .b(B), .c_in(C_IN), .a(A));
```

7.3 HIERARHIJSKA IMENA

- Projektovanje na više nivoa - dobija se složena struktura.
- Radi jednoznačnosti imena instanci (komponente) i nosioca podataka se dopune sa putanjom do njih.



NOSIOCI PODATAKA MODULI (INSTANCE)

Stimulus.q	**Stimulus.f1.Q**	
Stimulus.notq	**Stimulus.f1.NotQ**	**Stimulus.f1**
Stimulus.reset	**Stimulus.f1.R**	**Stimulus.f1.n1**
Stimulus.set	**Stimulus.f1.S**	**Stimulus.f1.n2**

8. HDL OPIS NA NIVOU LOGIČKIH KAPIJA (GATE-LEVEL MODELING)

- Drugi nivo apstrakcije - gate level modeling.
- Prvi nivo (modeliranje prekidačima) koriste samo proizvođači komponenti.
- Kola projektovana primenom tradicionalnih *SSI* i *MSI* komponenti se na ovaj način najlakše mogu preneti u svet *PLD*-ova.

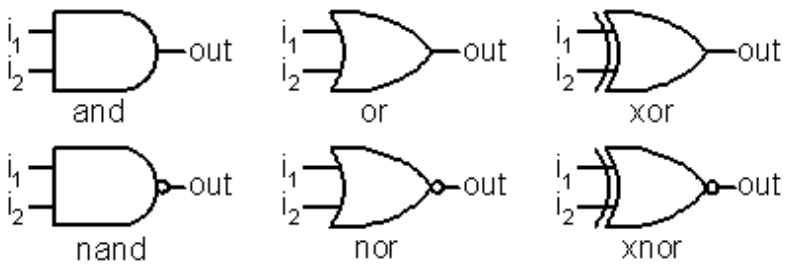
8.1 OPŠTE INFORMACIJE U VEZI VERILOG LOGIČKIH KAPIJA

- Logičke kapije su fabričke komponente (moduli) u jeziku Verilog HDL.
- Primena Verilog logičkih kapija se vrši instanciranjem, kao kod bilo kog modula.
- Imena logičkih kapija su predefinisana, treba uvek da se pišu na isti način da bi ih softver koji tumači Verilog opis mogao prepoznati.
- Na raspolaganju su razne I i ILI kapije i kola za sprezanje.
- Redovno se mogu instancirati i HDL opisi mnogih složenijih kola, ali ta mogućnost zavisi od primenjenog softvera.

8.1.1.a HDL *I* I *ILI* KAPIJE

- Jednobitni (skalarni) izlaz, dvo ili višebitni ulaz.
- U listi portova prvi elemenat je izlaz, ostali su ulazi.

Ime modula	Funkcija
and	I
nand	NI
or	ILI
nor	NILI
xor	isključivo ILI
xnor	isključivo NILI



8.1.1.b HDL *I* I *ILI* KAPIJE

Primeri instanciranja logičkih kapija:

```
wire OUT, IN1, IN2, IN3;
// Instanciranje osnovnih logičkih kapija sa imenima
and a1(OUT, IN1, IN2);
nand na1(OUT, IN1, IN2);
or or1(OUT, IN1, IN2);
nor nor1(OUT, IN1, IN2);
xor x1(OUT, IN1, IN2);
xnor nx1(OUT, IN1, IN2);
// Instanciranje NI kola sa tri ulaza
nand na1_3inp(OUT, IN1, IN2, IN3);
// Instanciranje I kola bez imena
// Dozvoljen način instanciranja logičkih kola
and (OUT, IN1, IN2);
```

8.1.1.c HDL *I* I *ILI* KAPIJE

- Proračun izlaznih vrednosti se vrši prema priloženim tabelama.
- Uzimaju se u obzir i neodređeni ulazi i ulazi sa stanjem velike impedanse.
- HDL logičke kapije uvek formiraju nizak ili visok logički nivo na svojim izlazima (nema međustanja), ali se dešava da se ne zna koji izlazni nivo će se pojaviti - zato ima x-eva u tabelama.

and	i1			
	0	1	x	z
i2 0	0	0	0	0
i2 1	0	1	x	x
i2 x	0	x	x	x
i2 z	0	x	x	x

nand	i1			
	0	1	x	z
i2 0	1	1	1	1
i2 1	1	0	x	x
i2 x	1	x	x	x
i2 z	1	x	x	x

or	i1			
	0	1	x	z
i2 0	0	1	x	x
i2 1	1	1	1	1
i2 x	x	1	x	x
i2 z	x	1	x	x

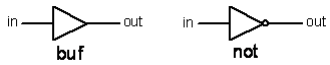
nor	i1			
	0	1	x	z
i2 0	1	0	x	x
i2 1	0	0	0	0
i2 x	x	0	x	x
i2 z	x	0	x	x

xor	i1			
	0	1	x	z
i2 0	0	1	x	x
i2 1	1	0	x	x
i2 x	x	x	x	x
i2 z	x	x	x	x

xnor	i1			
	0	1	x	z
i2 0	1	0	x	x
i2 1	0	1	x	x
i2 x	x	x	x	x
i2 z	x	x	x	x

8.1.2 OBIČNA KOLA ZA SPREZANJE

- Jednobitni ulaz.
- Jedno ili višebitni izlaz (grananje)
- Invertujuća i neinvertujuća verzija.



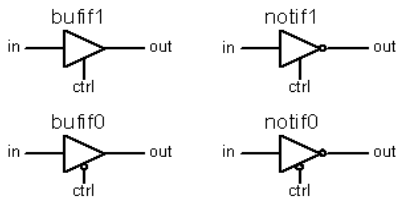
buf	
in	out
0	0
1	1
x	x
z	x

not	
in	out
0	1
1	0
x	x
z	x

```
// Instanciranje bafera i invertora sa jednim ulazom i izlazom
buf b1(OUT, IN);
not n1(OUT, IN);
 // Instanciranje sa više od jednog izlaza
buf b1_3out(OUT1, OUT2, OUT3, IN);
// Instanciranje invertora bez imena instance
// (dozvoljena metoda)
not(OUT1, OUT2, IN);
```

8.1.3 KOLA ZA SPREZANJE SA TRI STANJA

- Jednobitni ulaz za podatak i dodatni ulaz za kontrolu.
- Jedno ili višebitni izlaz (grananje)
- Invertujuća i neinvertujuća verzija.



bufif1	ctrl			
	0	1	x	z
in 0	z	0	x	x
in 1	z	1	x	x
in x	z	x	x	x
in z	z	x	x	x

notif1	ctrl			
	0	1	x	z
in 0	z	1	x	x
in 1	z	0	x	x
in x	z	x	x	x
in z	z	x	x	x

bufif0	ctrl			
	0	1	x	z
in 0	0	z	x	x
in 1	1	z	x	x
in x	x	z	x	x
in z	x	z	x	x

notif0	ctrl			
	0	1	x	z
in 0	1	z	x	x
in 1	0	z	x	x
in x	x	z	x	x
in z	x	z	x	x

```
// Instanciranje logičkih kola bufif
bufif1 b1(out, in, control);        // sa sopstvenim imenom
bufif0 (out, in, control);          // bez sopstvenog imena
// Instanciranje logičkih kola notif
notif1 n1(out, in, control);        // sa sopstvenim imenom
notif0 (out, in, control);          // bez sopstvenog imena
```

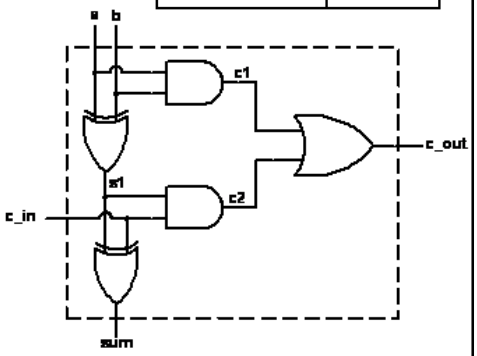
8.2.a PRIMER PROJEKTOVANJA NA NIVOU LOGIČKIH KAPIJA

- Cilj: HDL opis četvorobitnog potpunog sabirača.
- Prvo se definiše modul jednobitnog sabirača.

$$sum = (a \oplus b \oplus c\_in)$$
$$c\_out = a \cdot b + c\_in \cdot (a \oplus b)$$

c_in	a	b	sum	c_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

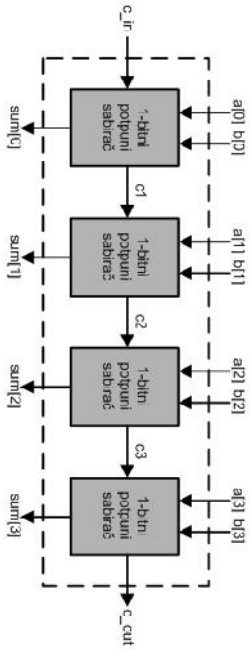
```
module FullAdder1bit(sum, c_out, a, b, c_in);
 // Deklaracije port-ova
output sum, c_out;
input a, b, c_in;
 // deklaracije internih veza
wire s1, c1, c2;
// Instanciranje logičkih kapija (bez imena)
xor (s1, a, b);
xor (sum, s1, c_in);
 and (c1, a, b);
and (c2, s1, c_in);
 or (c_out, c1, c2);
 endmodule
```



8.2.b PRIMER PROJEKTOVANJA NA NIVOU LOGIČKIH KAPIJA

Povezivanje četiri jednobitna potpuna sabirača.

```
module FullAdder4_bit(sum, c_out, a, b, c_in);
 // Deklaracija port-ova
output [3:0] sum;
output c_out;
 input [3:0] a, b;
input c_in;
 // Deklaracija internih veza (čvorova)
wire c1, c2, c3;
 // Instanciranja četiri jednobitna sabirača
FullAdder1bit fa0(sum[0], c1, a[0], b[0], c_in);
FullAdder1bit fa1(sum[1], c2, a[1], b[1], c1);
FullAdder1bit fa2(sum[2], c3, a[2], b[2], c2);
FullAdder1bit fa3(sum[3], c_out, a[3], b[3], c3);
 endmodule
```



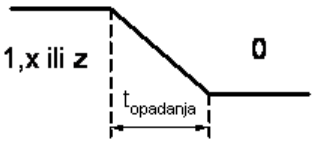
8.3.a KAŠNJENJA KOD HDL LOGIČKIH KAPIJA

- Radi modelovanja kašnjenja kod hardverskih logičkih kapija i HDL logičke kapije mogu imati kašnjenja.

Vrste kašnjenja:

- kašnjenje **porasta**

- kašnjenje **opadanja**



- kašnjenje **isključenja** - prelaz u stanje velike impedanse.
- prelaz **sa određene vrednosti na neodređenu vrednost** (x) - ne zadaje se posebno, već softver uzima najmanju vrednost od prethodne tri vrednosti.
- Uglavnom radimo sa **relativnim vremenima** (zadate vrednosti nisu u konkretnim jedinicama, na pr. ns).

8.3.b KAŠNJENJA KOD HDL LOGIČKIH KAPIJA - JEZIČKE KONSTRUKCIJE

Primenjuju se sledeće jezičke konstrukcije:

```
// Svi prelazi imaju jednako kašnjenje koje je određeno vrednošću
//parametra delay_time.
and #(delay_time) a1(out1, in1, in2);
 // Za kašnjenje porasta i opadanja su zadate različite vrednosti.
or #(rise_value, fall_value) o1(out2, in1, in2);
// Za kašnjenje porasta, opadanja i isključivanja su zadate tri različite
//vrednosti.
and #(rise_val, fall_value, turn_off_value) b1(out3, in, control);
```

Konkretni primeri:

```
and #(5) a1(out1, in1, in2);        // Svi prelazi imaju kašnjenje od 5
                                    // vremenskih jedinica.
and #(4,6) a2(out2, in1, in2);      // Vreme porasta = 4, vreme opadanja = 6.
bufif0 #(3,4,5) b1(out3, in1, in2);          // Vreme porasta = 3,
                                  // vreme opadanja = 4, vreme isključivanja = 5
```

8.3.1.a MININIMALNE, TIPIČNE I MAKSIMALNE VREDNOSTI KAŠNJENJA

- Kod hardverskih logičkih kapija razlikuju se kašnjenja čak i među kapijama unutar istog integrisanog kola određenog proizvođača.
- U HDL opisu zadaju se tri vrednosti kašnjenja za odgovarajuće kašnjenje (na pr. porast).
- Pri simulaciji bira se u softveru koju vrednost da uzima u obzir (na pr. minimalna vrednost kod svih kapija).
- Ako se za odgovarajuće kašnjenje zadaje samo jedno vreme, to se koristi kao tipična vrednost.

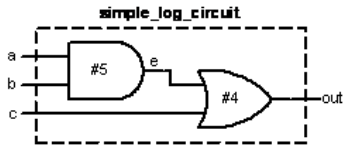
8.3.1.b MININIMALNE, TIPIČNE I MAKSIMALNE VREDNOSTI KAŠNJENJA - PRIMERI

```
and #(4:5:6) a1(out1, i1, i2);
// Vremena kašnjenja porasta, opadanja i isključivanja su jednaka:
// Min = 4, Tip = 5, Max = 6.
and #(3:4:5, 5:6:7) a2(out2, in1, in2);
// Vremena porasta:                    Min = 3, Tip = 4, Max = 5
// Vremena opadanja:                   Min = 6, Tip = 7, Max = 8
// Vremena isključivanja:              Min = 3, Tip = 4, Max = 5
// U slučaju kada su zadata samo vremena porasta i opadanja,
// vreme isključivanja se uzima
// kao minimalna vrednost od vremena porasta i opadanja
 and #(2:3:4, 3:4:5, 4:5:6) a3(out3, in1, in2);
// Vremena porasta:                    Min = 2, Tip = 3, Max = 4
// Vremena opadanja:                   Min = 3, Tip = 4, Max = 5
// Vremena isključivanja:              Min = 4, Tip = 5, Max = 6
```

8.3.2.a PRIMER ZA ANALIZU KAŠNJENJA

Prosto kombinaciono kolo:

Modul za opisivanje kola (sadrži kašnjenja):




```
module simple_log_circuit(out, a, b, c);
output out;
input a, b, c;
wire e;
and #(5) a1(e, a, b);    // Kašnjenje je 5 vremenskih jedinica
or #(4) o1(out, e, c);   // Kašnjenje je 4 vremenskih jedinica
endmodule
```

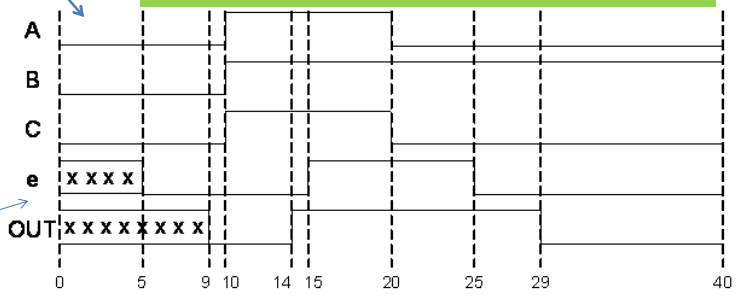
8.3.2.b A PRIMER ZA ANALIZU KAŠNJENJA

Ispitni modul:

Ulazni i izlazni signali:

```
module stimulus;
reg A, B, C;
wire OUT;
simple_log_circuit slc1(OUT, A, B, C);
initial
begin
                A = 1'b0; B = 1'b0; C = 1'b0;
        #10     A = 1'b1; B = 1'b1; C = 1'b1;
        #10     A = 1'b1; B = 1'b0; C = 1'b0;
        #20     $finish;
end
endmodule
```

Pre isteka kašnjenja izlazi pojedinih kola su neodređeni (x).



9. *HDL* OPIS NA NIVOU TOKA PODATAKA (DATAFLOW MODELING)

- Viši (treći) nivo.
- Projektant može da stavlja naglasak na operacije koje treba izvršiti nad podacima (efikasniji rad).
- I u ovom slučaju na kraju se dobija digitalno kolo ali sintezu ne vrši čovek nego softver.
- Za logičku sintezu danas su na raspolaganju mnogi efikasni softveri od proizvođača PLD-ova (uglavnom besplatno), i od nezavisnih softverskih kuća (uz plaćanje).
- Zbirno ime za projektovanje na trećem i četvrtom nivou je: *RTL - register transfer level* projektovanje.

9.1.a KONTINUALNA DODELA (CONTINUOUS ASSIGNMENT)

- Na ovaj način dajemo vrednost nosiocima podataka tipa *net*.
- Slično ponašanje kao kod logičkih kapija samo je lakše opisati složene operacije (viši nivo apstrakcije).
- Sve kontinualne dodele se navode sa ključnom reči ***assign***:

```
assign <kašnjenje> <dodela>;
```

- Nije obavezno zadati kašnjenje.

9.1.b A OSOBINE KONTINUALNIH DODELA

- Dodele se opisuju odgovarajućim izrazima.
- Svaki izraz sadrži jedan znak jednakosti.
- Na levu stranu znaka jednakosti piše se ime nosioca podatka tipa *net* (vektor ili skalar) (ne može ***reg***!).
- Na desnoj strani znaka jednakosti kombinacijom imena nosioca podataka *net* i/ili ***reg*** i odgovarajućih operatora se zadaje šta treba proračunati i dodeliti levoj strani.

```
// Kontinualna dodela.
// Podaci  out, in1 i in2 su tipa net.
assign out = in1 & in2;
// Kontinualna dodela vektorskim podacima tipa net
// addr1_bits i addr2_bits su 16-bitni vektorski podaci tipa reg.
assign addr[15:0] = addr1_bits[15:0] ^ addr2_bits[15:0];
```

9.1.1. IMPLICITNA KONTINUALNA DODELA

- Posebna (skraćena) metoda za pisanje kontinualnih dodela.
- Dodela se definiše pri deklaraciji nosioca podatka.
- Ne koristi se ključna reč ***assign*** u ovoj posebnoj konstrukciji.

```
//Regularan način pisanja kontinualne dodele,
// prvo se deklariše nosioc podatka.
wire out;
assign out = in1 + in2;

// Isti efekat se postiže implicitnom kontinualnom
dodelom.
wire out = in1 + in2
```

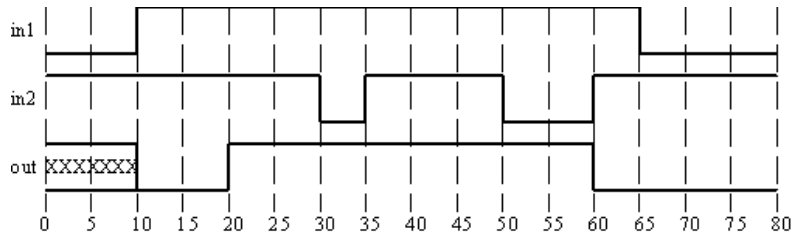
9.2 KAŠNJENJA U KONTINUALNIM DODELAMA

Tri metode za zadavanje kašnjenja

1. U pravilnim kontinualnim dodelama (pored ključne reči ***assign***)
2. U implicitnim dodelama.
3. Pri deklaraciji nosioca podatka tipa *net*.

9.2.1 KAŠNJENJA U PRAVILNIM KONTINUALNIM DODELAMA

Primer: `assign #10 out = in1 & in2;`



Šta vidimo?

- **Na početku** simulacije, do isteka kašnjenja izlaz je **neodređen**.
- **Promene** na ulazu prouzrokuju promene na izlazu tek **nakon isteka kašnjenja**.
- Skokovi/propadi u signalu (glitch) kraći od kašnjenja nemaju uticaja na izlaz.

9.2.1 KAŠNJENJA U IMPLICITNIM DODELAMA

- Kašnjenje se zadaje pri deklaraciji nosioca podatka:

```
wire #10 out = in1 & in2;
 // Gornji izraz ima isti efekat kao posebna
// deklaracija tipa podatka i
// pisanje izraza za kontinualnu dodelu
// sa kašnjenjem.
wire out;
assign #10 out = in1 & in2;
```

9.2.3 ZADAVANJE KAŠNJENJA PRI DEKLARISANJU NOSIOCA PODATKA TIPA net

- Kašnjenje se zadaje pri deklaraciji nosioca podatka tipa *net* (na pr. ***wire***).
- Pri korišćenju kontinualne dodele **logičko stanje se neće odmah pripisati** nosiocu podatka već nakon kašnjenja koje je definisano u ranije navedenoj deklaraciji.

```
wire #10 out;
assign out = in1 & in2;

// Gornji izrazi imaju isti efekat kao i sledeći:
wire out;
assign #10 out = in1 & in2;
```

9.3 IZRAZI, OPERANDI I OPERATORI

- Pri projektovanju na nivou toka podataka kolo opisujemo **matematičkim izrazima**.
- Izrazi se sastoje od operanada spojenih sa operatorima.
- Tri različita primera za izraze:

```
a ^ b
addr1[7:4] + addr2[7:4]
in1 | in2
```

9.3.2 OPERANDI

- Operandi su neki podaci/nosioci podataka (*net,* ***reg, integer, real***), konstante ili vrednosti funkcija (Ovde se ne bavimo Verilog funkcijama).
- U slučaju vektora operand može biti i jedan deo vektora (ne samo ceo vektor).

```
integer count, final_count;
final_count = count + 1;        // count je operand tipa integer
real a, b, c;
c = a – b;                      // a i b su operandi tipa real
reg [7:0] reg1, reg2;
reg [3:0] reg_out;
reg_out = reg1 [3:0] ^ reg2[3:0];
// reg1[3:0] i reg2[3:0] su operandi koji predstavljaju deo
// vektorskih podataka reg1 i reg2.
```

9.3.3 OPERATORI

- Pomoću operatora se definiše koje operacije treba izvršiti nad operandima (podacima).
- Primer:

```
d1 && d2        // && je binarni operator nad operandima d1 i d2.
!a[0]           // ! je unarni operator nad operandom a[0].
C >> 2  // >> je binarni operator nad operandima C i 2.
```

9.4 TIPOVI OPERATORA

Operacije u Verilog izrazima se dele na sledeće klase:

- aritmetičke,
- logičke,
- relacione,
- jednakosti,
- bit,
- redukcione,
- pomeračke,
- pridruživanje i umnožavanje,
- uslovne.

9.4.1 ARITMETIČKE OPERACIJE

- Sve operacije u tabeli imaju dva operanda (binarne operacije).
- + i - može da se odnosi i na samo jedan operand (unarne operacije).
- Negativne vrednosti treba dodeljivati samo nosiocima podataka tipa ***integer*** ili ***real***.

OPERACIJA	OPERATOR	BR.OP.
množenje	*	2
deljenje	/	2
sabiranje	+	2
oduzimanje	-	2
deljenje po modulu	%	2

```
A = 4'b0011; B = 4'b0100;      // Podatak A i B su vektorske veličine tipa reg
D = 6; E = 4;                  // Podaci D i E su tipa integer
 A * B                         // Množenje A i B. Rezultat je 4'b1100.
D / E                          // Deljenje D sa E. Rezultat je 1. Ostatak deljenja se odbacuje.
A + B                          // Sabiranje A i B. Rezultat je 4'b0111.
B – A                          // Oduzimanje A od B. Rezultat je 4'b0001.
in1 = 4'b101x;
in2 = 4'b1011;
 sum = in1 + in2;     // Rezultat sabiranja će biti: 4'bx
 // Operator deljenja po modulu daje ostatak pri deljenju dva broja.
13 % 3     // Rezultat je 1.
16 % 4     // Rezultat je 0.
```

9.4.2 LOGIČKE OPERACIJE

- Bez obzira na broj bita operand se smatra za logičku vrednost (jedan bit).
- Rezultat je uvek jedan bit (0,1).
- Kao operandi mogu da se koriste i razni izrazi.

OPERACIJA	OPERATOR	BR. OP.
I	&&	2
ILI	||	2
NE	!	1

```
A = 3; B = 0;
A && B                         // Rezultat je: ( 1 && 0 ) = 0.
A || B                         // Rezultat je: (1 || 0 ) = 1.
!A                             // Rezultat je: NE( 1 ) = 0.
!B                             // Rezultat je: NE( 0 ) = 1.
// Neodređeni bitovi u operandima.
A = 2'b0x; B = 2'b10;
A && B                         // Rezulat je: ( x && 1 ) = x.
 // Primena izraza kao operanda.
(a == 1 ) && ( b == 3 )        // Rezultat je 1 ako su oba uslova tačna.
// Rezultat je 0 ako je barem jedan od ta dva uslova netačan.
```

9.4.3 RELACIONE OPERACIJE

- Upoređuju se brojne vrednosti.
- Rezultat je jedan bit (0, 1, x).
- Ako je makar jedan bit operanada x ili z, rezultat je x.

OPERACIJA	OPERATOR	BR. OP.
veće	>	2
manje	<	2
veće ili jednako	>=	2
manje ili jednako	<=	2

```
A = 4; B = 2;
X = 4'b1010; Y = 4'b1011; Z = 4'b1xxx;

A <= B; // Rezultat je 0.
A > B;  // Rezultat je 1.
Y <= X; // Rezultat je 0.
Y < Z;  // Rezultat je x.
```

9.4.4 OPERACIJE JEDNAKOSTI

- Upoređuju se brojne vrednosti.
- Rezultat je jedan bit (0, 1, x).
- Ako je makar jedan bit operanada x ili z, rezultat je x.
- Kod *case* jednakosti rezultat je uvek određen.

OPERACIJA	OPERATOR	BR. OP.
jednako	==	2
nije jednako (različito)	!=	2
case jednako	===	2
case nejednako	!==	2

```
A = 4; B = 3;
X = 4'B1010; Y = 4'b1101;
Z = 4'b1xxz; M = 4'b1xxz; N = 4'b1xxx;

A == B    // Rezultat je 0.
X != Y    // Rezultat je 1.
X == Z    // Rezultat je x.
Z === M   // Rezultat je 1 jer svi bitovi se poklapaju uključujući x i z.
Z === N   // Rezultat je 0 jer bitovi sa najmanjom binarnom težinom se razlikuju.
M !== N   // Rezultat je 1.
```

9.4.5 OPERACIJE BIT PO BIT

- Suštinski se razlikuju od (običnih) logičkih operacija!
- Kod bit negacije negiramo bitove operanda jedan po jedan.
- Kod ostalih operacija operacije se vrše nad parovima odgovarajućih bitova operanada.

OPERACIJA	OPERATOR	BR. OP.
bit negacija	~	1
bit I	&	2
bit ILI	|	2
bit isključivo ILI	^	2
bit isključivo NILI	^~ ili ~^	2

```
X = 4'b1010;   Y = 4'b1101    ;        Z = 4'b10x1;
 ~X            // Negacija bit po bit. Rezultat je 4'b0101.
X & Y          // I operacija bit po bit. Rezultat je 4'b1000.
X | Y          // ILI operacija bit po bit. Rezultat je 4'b1111.
X ^ Y          // EX-ILI operacija bit po bit. Rezultat je 4'b0111.
X ^~ Y         // EX-NILI operacija bit po bit. Rezultat je 4'b1000.
X & Z          // I operacija bit po bit. Rezultat je 4'b10x0.
```

9.4.6 OPERACIJE REDUKCIJE

- Ima samo jednog operanda (unarna operacija).
- Operacija se primenjuje na uzastopne bitove operanda.
- Polazi se od bita najveće težine.

OPERACIJA	OPERATOR	BR. OP.
redukciono I	&	1
redukciono NI	~&	1
redukciono ILI	|	1
redukciono NILI	~|	1
redukciono isključivo ILI	^	1
redukciono isključivo NILI	^~ ili ~^	1

```
X = 4'b1010;
 &X            // Izračunava se pomoću jednačine: 1 & 0 & 1 & 0 = 1'b0.
|X             // Izračunava se pomoću jednačine: 1 | 0 | 1 | 0 = 1'b1.
^X             // Izračunava se pomoću jednačine: 1 ^ 0 ^ 1 ^ 0 = 1'b0.
// Redukcione operacije EX-ILI ili EX-NILI se mogu iskoristiti za generisanje
// bita parnosti ili neparnosti nekog vektora.
```

9.4.7 OPERACIJE POMERANJA

- Pomeraju se bitovi jednog vektora udesno ili ulevo za odgovarajući broj mesta.
- Prvi operand je sam vektor, drugi je broj mesta za koji se vrši pomeranje.
- Upražnjena mesta se popunjavaju nulama, istisnuti bitovi se gube.

OPERACIJA	OPERATOR	BR. OP.
pomeranje udesno	>>	2
pomeranje ulevo	<<	2

```
X = 4'b1100;
 Y = X >> 1;      // Y = 4'b0110. Sadržaj vektora X se pomera za jedan
// bit udesno i mesto najviše značajnog bita se popunjava sa 0.
Y = X << 1;      // Y = 4'b1000. Sadržaj vektora X se pomera za jedan
// bit ulevo i mesto najmanje značajnog bita se popunjava sa 0.
Y = X << 2;      // Y = 4'b0000. Sadržaj vektora X se pomera za dva bita
// ulevo i mesta dva najmanje značajnih bitova se popunjavaju sa 0.
```

9.4.8 OPERACIJA PRIDRUŽIVANJA

- Spaja više operanada (mogu biti vektori i skalari).
- Dužina vektorskih operanada treba da je unapred definisana.
- I delovi vektora mogu biti operandi.

OPERACIJA	OPERATOR	BR. OP.
pridruživanje	{}	proizvoljan broj

```
A = 1'b1;      B = 2'b00;      C = 2'b10;      D = 3'b110;

Y = { B, C };                  // Y = 4'b0010
Y = { A, B, C, D, 3'b001 };    // Y = 11'b10010110001
Y = { A, B[0], C[1] };         // Y = 3'b101
```

9.4.9 OPERACIJA UMNOŽAVANJA

- Prvi operand je jedna konstanta (broj), on određuje koliko puta treba napisati drugi operand jedan iza drugog.
- Umnožavanje se može kombinovati sa pridruživanjem.

OPERACIJA	OPERATOR	BR. OP.
umnožavanje	{{}}	2

```
reg A;
reg [1:0] B, C;
A = 1'b1;        B = 2'b00;        C = 2'b10;

Y = { 4{A} };                      // Y = 4'b1111
Y = { 4{A}, 2{B} };                // Y = 8'b11110000
Y = { 4{A}, 2{B}, C };             // Y = 10'b1111000010
```

9.4.10. USLOVNA OPERACIJA

OPERACIJA	OPERATOR	BR. OP.
uslovna	?:	3

- Ima tri operanda.
- Jezička konstrukcija:

uslov ? *izraz_za_slučaj_tačnog_uslova* : *izraz_za_slučaj_netačnog_uslova*;

- Ako je *uslov* tačan, koristi se *izraz_za_slučaj_tačnog_uslova*. Ako je *uslov* netačan, koristi se *izraz_za_slučaj_netačnog_uslova*.
- Ako je uslov neodređen (x) rezultantni vektor se sastoji iz poklapajućih bitova operanada i x-ova na mestima gde se ne poklapaju vrednosti.
- Radi kao multipleksor 2/1 ali može da se koristi i kao kolo za sprezanje sa tri stanja.

```
// Opisivanje multipleksora 2/1 pomoću uslovne operacije.
assign out = control ? in1 : in2;
// Modeliranje kola  za sprezanje sa tri stanja korišćenjem uslovne opercije.
assign addr_bus = drive_enable ? addr_out : 36'bz;
reg [1:0] A, B;    // Podaci A, B su reg tipa, veličine 2 bita.
 assign out = A[0] ? ( B[0] ? 1'b1 : 1'b0 ) : ( B[0] ? 1'b0 : 1'b1);
// Rekurzivno korišćenje uslovne operacije.
```

9.4.11 HIJERARHIJA OPERACIJA

- Najsigurnije je zagradama definisati redosled izvršavanja operacija.
- U suprotnom slučaju primenjuje se hijerarhija data u tabeli:

TIP OPERACIJE	OPERATOR	PRIORITET
unarna množenje, deljenje, deljenje po modulu	+, -, !, ~, *, /, %	najveći
sabiranje, oduzimanje pomeranje	+, -, <<, >>	
relacione jednakosti	< ,<=, >, >= ==, !=, ===, !==	
redukcione logičke	&, ~& ^, ~^ |, ~| &&, ||	
uslovne	?:	najmanji

10. *HDL* OPIS NA NIVOU PONAŠANJA (BEHAVIORAL MODELING)

- Ovo je najviši (četvrti) nivo apstrakcije.
- Ima formu algoritma, opisuje ponašanje digitalnog kola, ništa ne govori o realizaciji.
- Logičku sintezu (formiranje digitalnog kola) ne vrši projektant već određeni softver.
- Ispitne (simulacione) module skoro isključivo opisujemo na nivou ponašanja.

10.1. STRUKTUIRANE PROCEDURE (STRUCTURED PROCEDURES)

Opisi na nivou ponašanja se pišu u okviru dve struktuirane procedure. Na osnovu odgovarajućih ključnih reči te procedure se zovu:

1. ***initial*** i
2. ***always***

procedure.

- Jedan modul može da sadrži i više struktuiranih procedura.
- Ne može da krene opis nove procedure dok prethodni opis nismo završili.
- U toku simulacije i u realizovanom kolu procedure kreću u t=0 i izvršavaju se konkurentno (paralelno).

10.1.1.a PROCEDURA TIPA ***initial***

Jezička konstrukcija:

ključna reč ***initial*** + blok sa dodelama.

- Jedan modul može da sadrži jedan ili više ***initial*** procedura.
- Svaka procedura kreće u t=0, izvršavaju se i završavaju se nezavisno jedna od druge.
- Uloga: jednostruka nameštanja u digitalnim kolima.
- Najviše se koriste u simulacionim modulima (manje za razvoj kola).
- Ako je u proceduri samo jedna dodela, piše se odmah iza ključne reči.
- Ako ima više dodela u proceduri, oni se pišu između ključnih reči ***begin*** i ***end***.

10.1.1.b PROCEDURA TIPA *initial* -PRIMERI

- Modul koji sadrži više ***initial*** procedura:

```
module Stimulus;
reg a, b, m;
initial
        m=1'b0;  // U slučaju jedne dodele nisu potrebne ključne reči begin i end.
initial
begin
        #5 a=1'b1;          // U slučaju dve dodele već je potrebna primena ključnih reči
        #25 b=1'b0;         // begin i end. Kašnjenja u procedurama će se objasniti kasnije.
end
initial
        #50 $finish;        // Procedura tipa initial koja sadrži jednu instrukciju koja
                            //  definiše kraj simulacije.
endmodule
```

- Sve tri procedure kreću u t=0. Stvarna vremena izvršavanja su data u tabeli:

VREME	DODELA
0	m=1'b0
5	a=1'b1
30	b=1'b0
50	$finish

10.1. 2 PROCEDURA TIPA *always*

Jezička konstrukcija:
ključna reč ***always*** + blok sa dodelama.

- Zadate dodele se ponavljaju ciklički.
- I ovde se po potrebi koriste ključne reči ***begin*** i ***end***.
- Primer: definisanje takt signala primenom *always* procedure (simulacioni modul).

```
module clock_gen;
reg clock;
initial
        clock=1'b0;      // Takt signal kreće sa niskog logičkog nivoa.
always
        #10 clock=~clock;        // Posle svakih deset vremenskih jedinica
                         // treba invertovati logičku vrednost takt signala.
initial
        #1000 $finish;   // Procedura tipa initial - kraj simulacije.
endmodule
```

10.2 DODELE U *initial* I *always* PROCEDURAMA

- U dodelama na nivou ponašanja pripisuju se nove vrednosti nosiocima podataka tipa ***reg***, ***integer*** i ***real*** .
- Podatak se **ne menja dok ne dodelimo drugu vrednost** (memoriše se) - suštinska razlika u odnosu na dodele na nivou toka podataka.
- **Jezička konstrukcija** za definisanje dodela na nivou ponašanja:

<ime nosioca podatka> <znak dodele><izraz>

- Dva tipa: **blokirajuće** i **neblokirajuće** dodele.

10.2.1.a BLOKIRAJUĆE DODELE

- **Znak** blokirajuće dodele: **=** (znak jednakosti).
- Dodele se izvršavaju u onom redosledu kako su napisane u datoj (***initial*** ili ***always***) proceduri - aktuelna dodela blokira naredne dodele.
- Blokirajuće dodele napisane u posebnim (***initial*** ili ***always***) procedurama ne blokiraju jedan drugog, izvršavaju se nezavisno.

10.2.1.b BLOKIRAJUĆE DODELE

- Primer: deo jednog modula sa blokirajućim dodelama

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial
begin
        x=1'b0; y=1'b1; z=1'b1;
        count=0;
        reg_a=16'b0; reg_b=reg_a;
        #15 reg_a[2]=1'b1;
        #10 reg_b[15:13]={x,y,z};
        count=count+1;
end
```

REDOSLED	VREME IZVR.	DODELA
1	0	x=1'b0
2	0	y=1'b1
3	0	z=1'b1
4	0	count=0
5	0	reg_a=1'b0
6	0	reg_b=reg_a
7	15	reg_a[2]=1'b1
8	25	reg_b[15:13]={x,y,z}
9	25	count=count+1

10.2.2.a NEBLOKIRAJUĆE DODELE

- **Znak za navođenje** neblokirajuće dodele: **<=** (znak manje i znak jednakosti jedan pored drugog, isto kao operacija relacije).
- Aktuelna dodela ne blokira narednu dodelu, sve dodele dolaze na izvršavanje međusobno nezavisno.
- Ako nije navedeno kašnjenje, dodela se izvršava u t=0.
- Ne dolazi do sabiranja kašnjenja.

10.2.2.b NEBLOKIRAJUĆE DODELE

- Primer za mešovito korišćenje blokirajućih i neblokirajućih dodela:

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial
begin
        x=1'b0; y=1'b; z=1'b1;
        count=0;
        reg_a=16'b0; reg_b=reg_a;
        reg_a[2]<= #15 1'b1;
        reg_b[15:13]<= #10 {x,y,z};
        count<=count+1;
end
```

REDOSLED	VREME IZVR.	DODELA
1	0	x=1'b0
2	0	y=1'b1
3	0	z=1'b1
4	0	count=0
5	0	reg_a=1'b0
6	0	reg_b=reg_a
7	15	reg_a[2]<=1'b1
7	10	reg_b[15:13]<={x,y,z}
7	0	count<=count+1

10.2.2.c NEBLOKIRAJUĆE DODELE

- Neblokirajuće dodele mogu da spreče **efekat protrčavanja** (vrednost podataka zavisi od redosleda izvršavanja operacija, sam redosled je neizvestan).
- Metoda: u prvom koraku vrši se **privremeno memorisanje** podataka, u drugom koraku operacije se vrše sa memorisanim vrednostima.
- Ovaj način opisivanja predstavlja najbolje **okidanje na ivicu** karakteristično kod mnogih digitalnih kola.
- Primer:

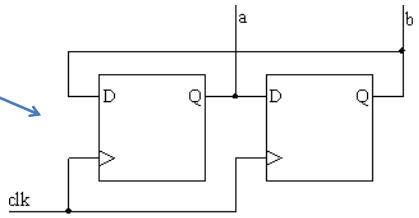
```
always @ (posedge clock)
begin
        reg1<=#1 in1;
        reg2<=@(negedge clock) in2^in3;
        reg3<=#1 reg1;
end
```

- Podaci na desnoj strani navedenih izraza se privremeno zapamte, zatim se izrazi izračunaju.
- Dodela se vrši ili odmah nakon proračunavanja ili kada dođe vreme za to (čeka se kašnjenje ili ivica takta).
- **Redosled upisivanja u registre je nebitan** jer se ne upisuje trenutna vrednost već ranije zapamćena vrednost.

10.2.2.d RAZLIKA IZMEĐU BLOKIRAJUĆIH I NEBLOKIRAJUĆIH DODELA

- Korišćenjem neblokirajućih dodela podaci se zamene pri svakoj uzlaznoj ivici takta.

```
always @ (posedge clock)
a<=b;
always @ (posedge clock)
b<=a;
```



- Korišćenjem blokirajućih dodela u oba registra će se upisati ista vrednost, na žalost ne zna se koja.

```
always @ (posedge clock)
a=b;
always @ (posedge clock)
b=a;
```

10.3 VREMENSKA KONTROLA DODELA U OPISIMA NA NIVOU PONAŠANJA

- Ako nema vremenske kontrole, nema napredovanja u vremenu pri simulaciji.
- Realan opis zahteva vremensku kontrolu.
- Sredstva koja se koriste za vremensku kontrolu:
  1. kašnjenje,
  2. okidanje na ivicu,
  3. okidanje na nivo.

10.3.1.a VREMENSKA KONTROLA ZADAVANJEM KAŠNJENJA

- Kašnjenje: vreme između dolaska dodele na izvršavanje i njenog stvarnog izvršavanja.
- Jezička konstrukcija za definisanje kašnjenja:

  # kašnjenje,

  gde se kašnjenje navodi kao broj, identifikator ili min/typ/max izraz.
- Primer za pisanje kašnjenja na pravilan način:

```
initial
begin
        #10 y=1'b1;
        #latency z=1'b0;
        #(4,5,6) q=1'b0;
end
```

10.3.1.b VREMENSKA KONTROLA KAŠNJENJIMA

Definisanje kašnjenja unutar dodela:

- Vrednost kašnjenja se piše na desnu stranu znaka dodele.
- Izraz na desnoj strani se izračuna u momentu dolaska na izvršenje (kod pravilnog pisanja kašnjenja i sâmo izračunavanje kasni).
- Dodela se odigrava kada istekne kašnjenje.

Definisanje kašnjenja unutar dodele →

```
initial
y= #5 x+z;
```

Isti rezultat se postiže privremenim memorisanjem podatka i pravilnim načinom pisanja kašnjenja. →

```
initial
begin
        temp_xz=x+z
        #5 y=temp_xz
end
```

10.3.2.a VREMENSKA KONTROLA PRIMENOM IVIČNOG OKIDANJA I OKIDANJA NA NIVO

- Većina savremenih digitalnih uređaja je sinhronog tipa (potreban je takt - clock).
- Događaji u digitalnim kolima mogu biti sinhronizovani na odgovarajući nivo ili odgovarajuću ivicu takt signala.
- Više se primenjuje okidanje na ivicu.
- Metode opisivanja sinhronizacije:
  1. pravilno ivično okidanje,
  2. čekanje na imenovani događaj,
  3. čekanje na promenu jednog signala iz skupa signala,
  4. čekanje na odgovarajući logički nivo.

PAŽNJA: U HDL OPISIMA NEMA GOTOVOG IZVORA TAKTA, PROJEKTANT TREBA DA KONSTRUIŠE I GENERATOR TAKTA I U OPISIMA HARDVERA I U OPISIMA SIMUALCIJE.

10.3.2.b OPISIVANJE PRAVILNOG IVIČNOG OKIDANJA

- Definiše se pomoću ključnih reči ***@, posedge*** i ***negedge***.
- Dodela se vrši ako je clock na logičkoj jedinici.
- Reaguje na uzlaznu ivicu.
- Reaguje na silaznu ivicu.
- Prvo se memoriše vrednost podatka *d*, sâmo dodeljivanje se dešava pri uzlaznoj ivici.

```
always @ (clock)
        q=d;
always @ (posedge clock)
        q=d;
always @ (negedge clock)
        q=d;
always
        q=@(posedge clock) d;
```

10.3.2.c VREMENSKA KONTROLA ČEKANJEM NA IMENOVANI DOGAĐAJ

1. Deklariše se jedan događaj (koristi se ključna reč ***event***).
2. Definiše se događaj.
3. Izvrši se dodela koja je čekala na nastupanje događaja.

```
event received_data;

always @(posedge clock)
begin
        if (last_data_packet)
                ->received_data;
end

always @ (received_data)
data_buf={data_pkt[0], data_pkt[1],
data_pkt[2], data_pkt[3]};
```

10.3.2.d ČEKANJE NA PROMENU JEDNOG SIGNALA IZ SKUPA SIGNALA - PREMA STANDARDU IZ 1995.

- Radi isto kao pravilno ivično okidanje samo ima više kontrolnih signala.
- Od više signala promena jednog (ivica!) pokreće dodelu.
- Između imena pojedinih signala piše se ključna reč ***or***.

```
always @(reset or clock or d)
begin
        if (reset)
                q=1'b0;
        else    if(clock)
                q=d;
end
```

- Na osnovu navedenog opisa sintetizuje se kolo sa okidanjem na nivo (D latch).
- I kod kola **sa okidanjem na nivo promene** se dešavaju **nakon ivice** kontrolnog signala (zato što se tada formira novi logički nivo), ipak se ne govori o okidanju na ivicu!

10.3.2.e ČEKANJE NA PROMENU JEDNOG SIGNALA IZ SKUPA SIGNALA - PREMA STANDARDU IZ 2001.

- Lista osetljivosti zadata u zagradi (kada treba obraditi dodele navedene u **always** bloku), može da se navede kao obično nabrajanje, **razdvojeno zarezima**.
- Ako želimo da sintetizujemo **kombinacionu** mrežu primenom **always** procedure, možemo da koristimo zapis **@(*)**.
- Značenje tog zapisa je da treba izvršiti dodele u svim situacijama kada se promeni bilo koja ulazna promenljiva.

```
always @(reset, clock, d)
begin
        if (reset)
                q=1'b0;
        else    if(clock)
                q=d;
end
```

```
always @(*)
begin
eSeg = 1;
if(~A & D) eSeg = 0;
if(~A & B & ~C) eSeg = 0;
if(~B & ~C & D) eSeg = 0;
end
```

10.3.2.f ČEKANJE NA ODGOVARAJUĆI LOGIČKI NIVO (OKIDANJE NA NIVO)

- Definiše se pomoću ključne reči ***wait***.
- Dodela će se odigrati kada je uslov na visokom logičkom nivou.
- U primeru se nakon svakih 20 vremenskih jedinica dešava jedno inkrementiranje, pod uslovom da smo to dozvolili visokim nivoom signala *count_enable*.

```
always
        wait (count_enable) #20 count=count+1;
```

10.4.a USLOVNE DODELE

- Kod ***initial*** i ***always*** procedura izvršavanje određene dodele može biti uslovljeno određenim uslovom.
- Pri opisivanju uslovljavanja koristimo ključne reči ***if*** i ***else.***
- Postoje tri slučaja:
  1. Koristimo samo ključnu reč ***if***.
  2. Koristimo ključne reči ***if*** i ***else***.
  3. Koristimo kombinaciju ***if – else if – else***.

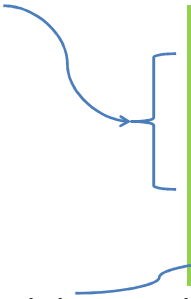
10.4.b USLOVNE DODELE - KORISTIMO SAMO KLJUČNU REČ *if*

- Navedena dodela će se izvršiti samo ako je tačan navedeni uslov.

```
if (!clock)        buffer=data;
```

10.4.c USLOVNE DODELE - KORISTIMO KLJUČNE REČI *if* I *else*

- Ove dodele će se izvršiti ako je tačan uslov naveden uz ključnu reč ***if***.



```
if (number_queued<MAX_Q_DEPTH)
begin
        data_queued=data;
        number_queued= number_queued+1;
end
else
        $display ("Queue full, Try again");
```

- Dodela navedena iza ključne reči ***else*** se izvršava ako uslov nije tačan.

10.4.d USLOVNE DODELE - KORISTIMO KOMBINACIJU KLJUČNIH REČI *if - else if - else*

- U ovom slučaju uslov može da ima više od dve vrednosti.
- Pri različitim vrednostima uslova treba izvršiti različite dodele.

```
if (alu_control==0)
        y=x+z;
else if (alu_control==1)
        y=x-z;
else if (alu_control==2)
        y=x*z;
else
        $diplay ("Invalid alu control signal")
```

10.5 VIŠESTRUKA GRANANJA

- Strukture ***if-else*** nisu pogodne kada uslov može da ima puno različitih vrednosti (opis neće biti pregledan).
- U takvim slučajevima bolje je koristiti ***case*** strukture. Za to se koriste ključne reči: ***case, casex, casez, endcase, default.***

10.5.1 ***case*** STRUKTURE

- Može se granati u tri pravca.
- Događaj za slučaj da nijedan uslov nije tačan. Ključna reč ***default*** se može pisati samo jednom u jednoj strukturi!
- Ostvarivanje multipleksora 4/1 primenom ***case*** strukture.

```
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 ("Invalid alu control signal");
endcase
```

```
module mux4_to_1(out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
case ({s1, s0})
            2'd0: out=i0;
            2'd1: out=i1;
            2'd2: out=i2;
            2'd3: out=i3;
            default: $display ("Invalid control signals");
endcase
endmodule
```

10.5.2 KORIŠĆENJE KLJUČNIH REČI *casex* I *casez*

- Cilj korišćenja ključnih reči casex i casez je skraćivanje ***case*** struktura.
- Pri korišćenju ključne reči ***casez*** bit uslova označen sa *z* može da ima bilo koju vrednost, neće imati uticaja na grananje.
- Pri korišćenju ključne reči ***casex,*** bit uslova označen sa *x* može da ima vrednost bilo *x* bilo *z*, neće imati uticaja na izvršavanje dodele.

```
reg[3:0] encoding;
integer state;
casex (encoding)
        4'b1xxx: state=3;
        4'bx1xx: state=2;
        4'bxx1x: state=1;
        4'bxxx1: state=0;
        default:  state=0;
endcase
```

10.6 PETLJE U OPISIMA NA NIVOU PONAŠANJA

- U ***always*** procedurama dodele se stalno ponavljaju.
- Definisanjem petlji ponavljanje dodela se može vezati za određeni uslov.
- Za definisanje raznih petlji koristimo ključne reči :

  while, for, repeat, forever.

10.6.1 PETLJA TIPA *while*

- Jezička konstrukcija: ***while*** (izraz)
- Dodele unutar petlje se ponavljaju sve dok izraz u zagradi (uslov) daje tačnu vrednost.

```
integer count;
initial
begin
        count=0;
        while (count < 127)
        begin
                $display ("count=%d", count);
                count=count+1;
        end
end
```

10.6.2 PETLJA TIPA *for*

- Jezička konstrukcija: ***for*** (izraz)
- Izraz u zagradi sadrži tri stvari:
  1. početni uslov,
  2. ispitivanje uslova završetka,
  3. promena uslova koja se ispituje u vezi završetka.
- Prvenstveno se koristi za definisanje brojača.

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

10.6.3 PETLJA TIPA *repeat*

- Jezička konstrukcija: ***repeat*** (brojna vrednost)
- Koristi se kada znamo unapred koliko puta treba izvršiti dodelu unutar petlje.

```
integer count;
initial
begin
        count=0;
        repeat (128)
        begin
                $display (“count=%d”, count);
                count=count+1;
        end
end
```

10.6.4 PETLJA TIPA *forever*

- Jezička konstrukcija: ***forever***
- Nema uslova.
- U principu ponavlja se do beskonačnosti.
- Pogodan je za generisanje signala takta (clock).

```
initial
begin
        clock=1’b0;
        forever #10 clock=~clock;
end
```

10.7 REDNI I PARALELNI BLOKOVI

- Blokovi služe za grupisanje dodela.
- Redni blok: definiše se pomoću ključnih reči ***begin, end***. Navedene dodele se izvršavaju u redosledu pisanja.
- Paralelni blok: definiše se pomoću ključnih reči ***fork, join***. Dodele dolaze na izvršavanje istovremeno. Stvarno vreme izvršavanja se može dodatno kontrolisati navođenjem kašnjenja.

10.7.1 REDNI BLOKOVI

- Dodele dolaze na izvršavanje po redu navođenja.
- Neće se pokrenuti nova dodela dok se prethodna nije izvršila.
- Kašnjenja zadata kod pojedinih dodela su relativna, računaju se u odnosu na momenat izvršenja prethodne dodele (sabiraju se kašnjenja).

Bez kašnjenja

```
initial
begin
        x=1'b0;
        y=1'b1;
        z={x,y};
        w={y,x};
end
```

Sa kašnjenjem

```
reg x,y;
reg [1:0] z, w;
initial
begin
        x=1'b0;
        #5 y=1'b1;
        #10 z={x,y};
        #20 w={y,x};
end
```

10.7.2 PARALELNI BLOKOVI

- Dodele dolaze na izvršavanje kao kod neblokirajućih dodela ali se ne vrši prethodno memorisanje ulaznih vrednosti.
- Navedena kašnjenja su apsolutne vrednosti (ne nadovezuju se).
- Nastaje efekat protrčavanja ako u istom bloku modifikujemo vrednost podatka i odmah ga i koristimo u drugoj dodeli.

Sa kašnjenjem

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

Bez kašnjenja, pojavljuje se efekat protrčavanja

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

10.7.3 KOMBINOVANI BLOKOVI

- Redni i paralelni blokovi se mogu kombinovati.
- U navedenom primeru redni blok je glavni. Dodele rednog bloka se izvršavaju uzastopno.
- Paralelni blok se izvršava kada dođe red na njega unutar rednog bloka.

```
initial
begin
        x=1'b0;
        fork
                #5 y=1'b1;
                #10 z={x,y};
        join
        #20 w={y,x};
end
```

10.7.4 IMENOVANI BLOKOVI

- Blokovima možemo dati ime.
- Kod imenovanih blokova:
  1. Možemo deklarisati lokalne nosioce podataka.
  2. Pojedinim nosiocima podataka možemo pristupiti (pri simulaciji) preko hijerarhijskih imena.
  3. Možemo prekinuti izvršenje.


```
module top;

initial
begin: block1
integer i;
.....
end

initial
fork: block2
reg i;
.....
join
endmodule
```

10.7.4 PREKIDANJE IMENOVANOG BLOKA

- Primenom ključne reči ***disable*** možemo prekinuti imenovani blok.
- U navedenom primeru izlazimo iz petlje ***while*** ako je naredni bit vektora *flag* jedinica.

```
initial
begin
        flag=16'b0010_0000_0000_0000;
        i=0;
        begin: block1
        while (i<16)
                begin
                        if (flag[i])
                        begin
                                $display ("True bit at element number %d", i);
                                disable block1;
                        end
                        i=i+1;
                end
        end
```

Kraj III. dela

PROJEKTOVANJE PRIMENOM
PROGRAMABILNIH LOGIČKIH KOLA (*PLD*)
HARDVERSKI JEZIK *VERILOG*